4

Слон в посудной лавке. Внедрение ошибок

Внедрение ошибок — это наука и в то же время искусство обходить механизмы безопасности, вызывая в устройстве небольшие аппаратные поломки прямо во время его нормальной работы. Внедрение ошибок может повлечь за собой больший риск для безопасности системы, чем анализ побочных каналов. Анализ побочных каналов выполняется для поиска криптографических ключей, а внедрение ошибок позволяет атаковать другие механизмы безопасности, например безопасную загрузку, которая, помимо обеспечения полного контроля над системой, дает возможность достать ключи непосредственно из памяти, не прибегая к анализу побочного канала.

Внедрение ошибок вынуждает устройство работать за пределами его нормальных рабочих параметров, а желаемый результат достигается путем манипуляций с физическими параметрами. В этом заключается основное различие между «естественными ошибками» и «ошибками, созданными атакующими». Атакующие сами проектируют ошибки, чтобы точечно отключить сложные системы или вызывать определенную реакцию устройства, которая позволит им обойти механизмы безопасности. Результат может быть разным: от повышения привилегий до извлечения секретного ключа.

Высокая точность внедрения ошибок напрямую зависит от точности спроектированного устройства для внедрения. Менее точные устройства порождают более неожиданные эффекты, которые, ко всему прочему, будут при каждом

запуске разными, а это означает, что лишь некоторые из созданных ошибок будут использоваться для дела. Атакующие пытаются свести к минимуму количество попыток внедрения ошибок, чтобы как можно дольше продлить время эксплуатации. В главе 5 мы рассмотрим методы внедрения ошибок, а также понаблюдаем за тем, что происходит с микросхемой при возникновении ошибки.

Внедрение ошибок на практике не всегда уместно, так как для внедрения обычно требуется физический доступ к цели. Если целевое устройство надежно хранится в закрытой серверной комнате, то внедрить ошибки не получится. Если логические аппаратные и программные атаки не принесли желаемого результата, но физический доступ к цели у вас есть, то внедрение ошибок может спасти ситуацию. (Внедрение ошибок, вызванных программным обеспечением, не считается, поскольку для внедрения такой ошибки физический доступ не требуется. Подробнее об этом сказано в подразделе «Программные атаки на аппаратные средства» раздела «Типы атак» в главе 1.)

В этой главе мы обсудим основы внедрения ошибок и обоснования применения этого метода в целом. Мы также выполним тренировочное исследование реальной библиотеки (OpenSSH) и попробуем обойти механизмы аутентификации с помощью ошибок. На практике ошибки непредсказуемы, и перед внедрением необходимо произвести тщательную настройку параметров тестового стенда, поэтому мы также исследуем различные части стенда для внедрения неисправностей и стратегии настройки параметров.

Внедрение ошибок в механизмы безопасности

Устройства могут иметь несколько механизмов безопасности, с которыми можно поэкспериментировать путем внедрения ошибок. Например, функция отладки порта JTAG может быть доступна лишь после ввода пароля, в прошивке устройства может храниться цифровая подпись, а в аппаратном обеспечении ключ может храниться в надежном месте, скрытом от программной части. Любой здравомыслящий аппаратный инженер для обозначения состояния *«доступ разрешен»* или *«доступ запрещен, я вас не звал, уходите»* будет использовать один бит. Предполагается, что значение в данном бите будет сохраняться до тех пор, пока его программный контроллер не даст указание на изменение.

Поскольку на практике внедрение ошибок работает довольно хаотично, сложно атаковать именно тот бит, на котором держится весь механизм безопасности. Предположим, у вас есть устройство для внедрения ошибок, которое переключает один конкретный бит в определенный момент времени (эдакий единорог в мире внедрения ошибок: он красив, и все хотят себе такого, но на практике его не существует, если не считать микрозондирования, но оно уже из другой области).

Итак, мы можем использовать внедрение ошибок для обхода различных механизмов безопасности. Например, когда устройство загружается и выполняет проверку подписи прошивки, мы можем инвертировать логическое значение, отвечающее за *действительность подписи*. Мы также можем инвертировать биты блокировки на заблокированных функциях, таких как криптографические механизмы, ключ от которых мы знать не должны. Можно даже инвертировать биты прямо во время выполнения криптографического алгоритма, чтобы восстановить материал криптографического ключа. Рассмотрим некоторые из этих механизмов безопасности более подробно.

Обход проверки подписи прошивки

Современные устройства часто загружаются с образов прошивки, хранящихся во флеш-памяти. Чтобы запретить загрузку со взломанных образов прошивки, производители устройств подписывают их цифровой подписью, которая хранится где-то рядом с образом прошивки. При загрузке устройства образ и подпись проверяются с помощью открытого ключа, связанного с производителем устройства. Когда подпись проходит проверку, устройство может загрузиться. Проверка защищена криптографически, но в итоге устройство принимает бинарное решение: либо загружаться, либо нет. В загрузочном программном обеспечении устройства это решение обычно сводится к инструкции условного перехода. Нацелив внедрение ошибки на этот условный переход, можно сделать проверку пройденной, даже если образ был подменен. Даже если ПО является достаточно сложным, контролируемая ошибка в одном месте может обрушить всю безопасность, как карточный домик.

Доступ к устройству во время его загрузки позволяет злоумышленнику скомпрометировать любое загруженное программное обеспечение, а это может быть операционная система или любые другие приложения, которые общаются с нужными вам компонентами устройства.

Получение доступа к заблокированным функциям

Безопасная система должна контролировать доступ к своим функциям и ресурсам. Например, приложение не должно иметь доступа к памяти другого приложения, к механизму DMA[1] должно иметь доступ только ядро, а доступ к файлу должен иметь только авторизованный пользователь.

При несанкционированной попытке доступа к ресурсу устройство проверяет определенный бит (или биты) управления доступом и говорит, что «доступ запрещен». Решение часто принимается по состоянию одного бита и осуществляется

[1] DMA (direct memory access) — прямой доступ к памяти. — *Примеч. науч. ред.*

одной командой условного перехода. Идеальное устройство внедрения ошибок атакует именно эту точку отказа и может инвертировать бит. Как результат — достижение разблокировано.

Восстановление криптографических ключей

Ошибки, которые появляются в ходе выполнения криптографических процессов, могут привести к утечке криптографического ключа. По этой теме написано множество работ, обычно в рамках темы *дифференциального анализа неисправностей* (differential fault analysis, DFA). Это название происходит от использования дифференциального анализа ошибочного выполнения шифра, в ходе которого сравниваются различия между правильными и ошибочными выводами шифра. Существуют известные атаки DFA на криптографические технологии AES, 3DES, RSA-CRT и ECC.

Обычно суть атаки на криптографические алгоритмы состоит в том, чтобы выполнить расшифровку известных входных данных, иногда без внедрения ошибок. Анализ выходных данных может позволить определить сам ключ. В известных DFA-атаках на 3DES для получения полного ключа использовалось менее 100 ошибок. Для AES достаточно одной-двух. Больше информации доступно в статье *Information-Theoretic Approach to Optimal Differential Fault Analysis* Кадзуо Сакиямы и др. Классическая атака Bellcore на RSA-CRT выполняется за одну ошибку, которая позволяет извлечь весь закрытый ключ RSA, какой бы длины он ни был. Алгоритм атаки кажется каким-то волшебством, даже если вы разберетесь в его математике! Подробнее об этом можно прочитать в работе *Fault Analysis in Cryptography* (Springer, 2012 г.), вышедшей под редакцией Марка Джоя и Майкла Танстолла.

Можно провести не-DFA-атаки на криптографический алгоритм, подделав реализацию шифра так, чтобы выполнялся лишь один раунд шифрования без добавления ключей, частичного обнуления ключей или других искажений. Все эти методы требуют некоторого анализа криптографических свойств алгоритма и ошибки, поскольку без этого не удастся понять, как в результате ошибочного выполнения извлечь ключ. В самом тривиальном случае можно получить дампы памяти, содержащие ключевой материал. Мы вернемся к DFA в эксперименте в главе 6.

Упражнение по внедрению ошибок в OpenSSH

Рассмотрим, как внедрить ошибку в момент доступа к устройству через соединение OpenSSH и определить возможные точки внедрения в реальном коде, отвечающем за безопасность. Предположим, что на устройстве отключены порты проверки подлинности прошивки и отладки, а интерфейс для общения с ним — порт Ethernet, подключенный к слушающему серверу OpenSSH.

Внедрение ошибок в код C

Чтобы внедрить ошибку на этапе ввода пароля, рассмотрим код OpenSSH 7.2p2, приведенный в листинге 4.1.

Листинг 4.1. Код проверки пароля OpenSSH в файле auth2-password.c

```
--пропуск--
50
51 int userauth_passwd(Authctxt *authctxt)
52 {
53         char *password, *newpass;
54         int authenticated = 0;
55         int change;
56         u_int len, newlen;
57
58         change = packet_get_char();
59         password = packet_get_string(&len);
60         if (change) {
61                 /* discard new password from packet */
62                 newpass = packet_get_string(&newlen);
63                 explicit_bzero(newpass, newlen);
64                 free(newpass);
65         }
66         packet_check_eom();
67
68         if (change)
69                 logit("password change not supported");
70         else if (PRIVSEP(auth_password(authctxt, password)) == 1)
71                 authenticated = 1;
72         explicit_bzero(password, len);
73         free(password);
74         return authenticated;
75 }
--пропуск--
```

Функция `userauth_passwd`, приведенная в листинге 4.1, явно отвечает за правильность пароля. Переменная `authenticated` в строке 54 хранит информацию о том, разрешен ли доступ. Прочтите этот код и подумайте, как с помощью ошибок повлиять на его выполнение так, чтобы переменная `authenticated` стала равна `1`, если предоставлен неверный пароль. Предположим, вы можете инвертировать биты или отправлять код по другой условной ветви. Подумайте и найдите хотя бы три способа; затем прочитайте следующие ответы.

Приведем несколько способов внедрить в этот код ошибку:

- задать ненулевое значение флагу `authenticated` в строке 54 или после нее;
- изменить возвращаемое значение функции `auth_password()` в строке 70 на `1`;
- изменить результат сравнения в строке 70 на `true`;

- в строке 70 выполнять сравнение с другим, подложным паролем;
- запросить смену пароля, задать `change=1`, затем сделать так, чтобы переменная `newpass` в строке 62 указывала на тот же адрес, что и `password`, а затем использовать два вызова функции `free`, которая теперь освобождает одну и ту же память в строках 64 и 73.

Последний сценарий не вполне реалистичен, поскольку такой контроль над целью не встречается на практике. Но остальные ошибки вполне базовые. Если вы отследите код, ведущий к функции `auth_password()`, то найдутся еще десятки возможностей для ошибок.

Важный момент заключается в том, что некоторые ошибки на практике внедряются легче других. Как правило, чем точнее синхронизация или конкретнее требуемый эффект, тем ниже вероятность успешного внедрения.

Внедрение ошибок в машинный код

Проанализировать код на языке C полезно в качестве упражнения, но ЦП работает не на нем. ЦП выполняет инструкции машинного кода, созданные из кода C. Машинный код людям читать тяжело, поэтому мы рассмотрим код на ассемблере, который является прямым представлением машинного кода. Инструкции кода ассемблера находятся на более низком уровне абстракции, чем C, и напрямую описывают действия, происходящие внутри устройства (на высокопроизводительных процессорах есть еще один, более низкий уровень абстракции, но он обычно скрыт, поэтому мы не будем рассматривать его).

Сбои возникают внутри оборудования на физическом уровне и уже оттуда распространяются на более высокие уровни абстракции. Внутри ЦП может произойти инверсия бита, когда он выполняет некий исполняемый файл, а тот создается из некоего исходного кода. Таким образом, несмотря на связь между ошибкой и предшествующим кодом C, анализируя код именно на ассемблере, мы окажемся на один слой ближе к ошибке. Дополнительную информацию по этому вопросу можно найти в статье *Fault Attacks on Secure Embedded Software: Threats, Design and Evaluation* Билгидея Юса и др.

Для этой книги мы взяли бинарный файл OpenSSL и загрузили его в дизассемблер IDA Pro. Ассемблерный код для рассмотренной нами части функции `userauth_passwd` приведен на рис. 4.1.

По соглашению, функция возвращает данные о состоянии аутентификации пользователя в регистр `rax`. Чтобы программа прочитала значение `authenticated == true`, регистр `rax` должен быть ненулевым. Обратите внимание, что `eax` — это всего лишь младшие 32 бита `rax`, и это наводит на мысль об условиях, которые обнуляют регистр `rax`. Для этого посмотрим на последний блок с меткой `loc_24723` (отмечен цифрой ❶). Дальше пойдут спойлеры.

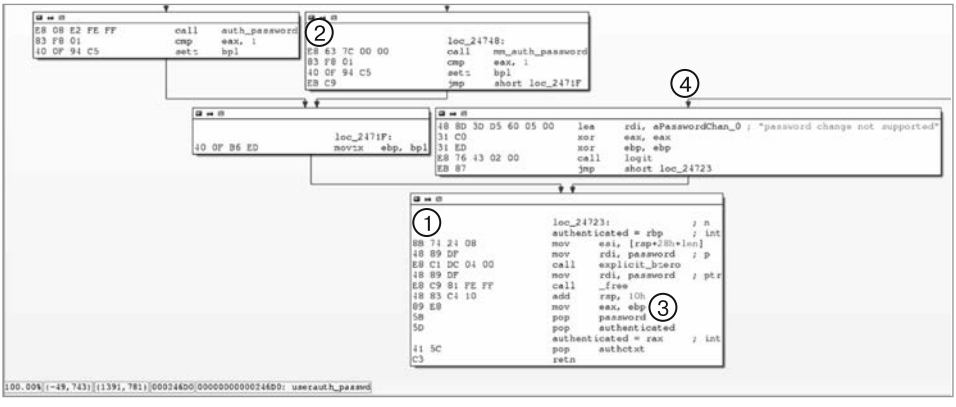

Рис. 4.1. Поиск инструкций, которые должны отработать с ошибкой, в ассемблерном коде

Нужно, чтобы на момент входа в последний блок `loc_24723` в точке ❶ выполнялось условие `ebp != 0`. В ассемблере Intel `ebp` — это младшие 32 бита регистра `rbp`, а `bpl` — младшие восемь бит `rbp/ebp`. Теперь вернитесь к коду и подумайте, как сделать так, чтобы выполнялось `ebp != 0`, с помощью ошибки, которая инвертирует бит или пропускает инструкцию. Снова сделаем паузу.

Мы нашли несколько способов.

- В `loc_24748` (пункт ❷) пропустить вызов функции `mm_auth_password` и понадеяться, что в `eax` было значение `1`. Если да, то инструкция `setz bpl` приводит к `ebp != 0`, и поэтому `authenticated == true`.
- В `loc_24748` (пункт ❷) с помощью внедрения ошибки пропустить инструкцию `cmp eax,1`, а затем понадеяться на то, что `auth_password` установила флаг `z` равным `1`.
- Этот способ вы, наверное, не найдете, если не проанализируете вызов функции в бинарном формате (всегда смотрите на общую картину, ибо ошибки находятся именно там!). После вызова `auth_password` переменная `authenticated` появляется в `eax`, затем в флаге `bpl`, затем `ebp` и, наконец, в `rax` (например, в пункте ❸ выполняется копирование из `ebp` в `rax/eax`), что означает, что вы можете вызвать ошибку в любом месте этой цепочки в соответствующем регистре, чтобы установить для `authenticated` значение `1`.
- Установить флаг смены пароля `change` равным `true` (по протоколу или с помощью ошибки. Обратите внимание, что любое ненулевое значение оценивается как истинное), что дает ответ `password change not supported` в пункте ❹ в вызове функции `logit`. Внедрите ошибку, чтобы пропустить шаг `xor ebp,ebp` и надейтесь, что в `ebp` не ноль.

В коде ассемблера есть много мест, куда можно внедрить ошибку. Вам не нужно чересчур вдумчиво выбирать точку внедрения, которая позволит достичь определенного результата. В этом примере мы рассмотрели много ошибок, которые могут установить `authentication == true` и обойти проверку пароля.

OpenSSH разрабатывался без учета внедрения ошибок, и в модель угроз эта атака не входит. В главе 14 вы узнаете, как применить в программном обеспечении контрмеры, позволяющие снизить эффективность внедрения неисправностей. В этой же главе вы найдете информацию об *имитации ошибок*. Этот метод позволяет определить, насколько хорошо код может противостоять ошибкам. Повышение устойчивости кода к естественным ошибкам также в некоторой степени защищает от вредоносных ошибок, но не полностью. Дополнительную информацию о внедрении ошибок без злонамеренных действий можно найти в книге *Software Fault Injection* (Wiley, 1998 г.) Джеффри Воаса и др. О том, как меры безопасности в микросхемах не всегда превращаются в механизмы безопасности, можно почитать в работе *Safety ≠ Security: A Security Assessment of the Resilience Against Fault Injection Attacks in ASIL-D Certified Microcontrollers* Нильса Вирсма и др. Предыдущие примеры исходного кода и ассемблерного кода показывают, как одиночный сбой, например обход пароля, может серьезно повлиять на безопасность.

Тот самый слон

До сих пор мы предполагали, что у нас есть мифическое идеальное однобитовое устройство внедрения ошибок, которое мы назвали нашим *единорогом* внедрения ошибок. К сожалению, такового не существует, поэтому посмотрим, насколько близко мы сможем подобраться к нашему мифическому единорогу, имея в распоряжении лишь реальные земные инструменты. На практике лучшее, на что мы можем надеяться, — это возможность хотя бы время от времени вызывать полезные ошибки. Самый простой способ внедрить ошибку — увеличить или понизить напряжение в цепи и перегреть ее. Существуют и околофантастические методы, например воздействие с помощью сильных электромагнитных импульсов, сфокусированных лазерных импульсов или альфа-частиц либо гамма-лучей.

Атакующий выбирает средства внедрения ошибок, а затем настраивает их время, продолжительность и другие параметры, чтобы максимизировать эффективность атаки и достичь своей цели. Цель защитника — свести к минимуму эффективность атаки, в которой внедрение ошибок увенчалось успехом.

В реальной задаче вы не сможете внедрить идеальную ошибку с первой попытки, поскольку ее параметры вам поначалу неизвестны. Если бы вы знали правильные параметры, то ваш единорог воздействовал бы на цель точечно и нужном месте.

Однако поскольку устройство внедрения всегда вносит некоторые временные отклонения, процесс будет сопровождаться несколькими разными эффектами, причем даже при использовании одних и тех же настроек. На практике неточность устройства внедрения делает ошибку более случайной, и тогда вам потребуется несколько попыток, чтобы провести нужную атаку.

Чтобы решить эту дилемму, вам нужно создать систему, в которой вы могли бы экспериментировать с ошибками и пытаться сделать цель максимально точной. Идея состоит в том, чтобы запустить целевую операцию, дождаться сигнала триггера, указывающего, что целевая операция выполняется, внедрить ошибку, зафиксировать результаты и при необходимости сбросить цель в исходное состояние для новой попытки.

Целевое устройство и цель ошибки

Как мы уже упоминали, внедрение ошибок требует физического контроля над устройством, поэтому сначала вам нужно собственно устройство (а лучше несколько, на случай если вы что-то поджарите). Лучше выбрать простое устройство, например Arduino или другой медленный микроконтроллер, для которого вы уже написали некий код.

Затем вам нужно понять цель, ради которой затевается внедрение ошибки, например обход проверки пароля. Выше вы уже видели анализ кода OpenSSH как на языке C, так и на ассемблере, и мы нашли множество способов достижения цели. Имейте в виду, что C, ассемблер и Verilog или VHDL — просто способы представления того, что происходит с физическим оборудованием. Здесь вы пытаетесь манипулировать оборудованием, вмешиваясь в его физическую среду. Поступая так, вы нарушаете предположения, которые делают инженеры, например о том, что транзистор переключается только по указанию, логический вентиль переключится до следующего тактового импульса, инструкция процессора будет выполнена правильно, переменные в программе C будут хранить свое значение, до тех пор пока не будут перезаписаны, или что арифметическая операция всегда будет вычисляться правильно. Вы вызываете ошибку на физическом уровне, чтобы достичь целей на более высоком уровне.

Инструменты внедрения ошибок

Чем лучше вы понимаете физику, тем лучше вам удастся спланировать устройство для внедрения ошибок. Но в то же время докторская степень по физике вам не нужна. В главе 5 мы более подробно поговорим о физике, лежащей в основе различных методов внедрения, и о конструкции устройства внедрения ошибок.

Устройство внедрения, генерирующее тактовый сигнал для целевого устройства, может копировать обычный тактовый сигнал этого устройства, но в какой-то момент добавлять лишний очень быстрый такт для ускорения процесса. Цель

введения быстрого такта состоит в том, чтобы вызвать сбой в ЦП. На рис. 4.2 показано, как выглядит подобный тактовый сигнал.

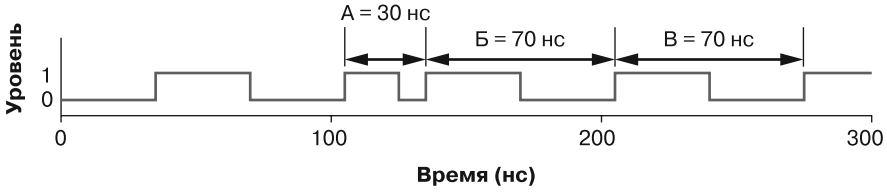

Рис. 4.2. Провокация сбоя ЦП с помощью быстрого цикла

На графике виден обычный тактовый сигнал с периодом 70 нс, но затем наступает цикл А. Он прерывается, и цикл Б начинается только спустя 30 нс после начала цикла А. Продолжительность Б и В снова составляет 70 нс. Это может вызвать сбой в работе чипа во время цикла А и/или Б.

Наносекундные сдвиги при работе с тактовыми частотами порядка гигагерц очень важны, поскольку на частоте 1 ГГц одна наносекунда составляет продолжительность полного тактового цикла. Достижение подобной точности синхронизации на практике означает необходимость создания особых аппаратных схем для внедрения ошибок.

ПРИМЕЧАНИЕ

Что касается примера с тактовым сигналом, то в подразделе «Поиск эффективных ошибок» далее в главе мы рассмотрим схему, которая имитирует точный тактовый сигнал, отсчитывает целевой такт, а затем отправляет его в цепь, чтобы внедрить неисправность. Здесь вам очень пригодятся программируемые вентильные матрицы (field-programmable gate arrays, FPGA) и более быстрые микроконтроллеры.

Вам нужно научиться контролировать как можно больше точек внедрения ошибок, поэтому ваше устройство внедрения должно быть программируемым. Чтобы найти правильные параметры ошибок, потребуется множество экспериментов, каждый со своими настройками. В примере с внедрением в тактовый сигнал нам нужна возможность задать устройству внедрения исходную тактовую частоту, частоту разгона и момент внедрения. Таким образом, проведя серию экспериментов, вы научитесь контролировать частоту внедрений и поймете, какие именно настройки вызывают аномалию или повторяющийся эффект.

Подготовка целевого устройства и мониторинг

Процесс подготовки внедрения ошибок зависит от вашей цели и типа ошибки, которую вы собираетесь внедрить. Основные действия одни и те же: отправка команды на целевое устройство, получение результатов от целевого устройства,

управление сбросом целевого устройства, управление триггером, отслеживание реакции целевого устройства и формирование модификаций, характерных для неисправности. На рис. 4.3 показана схема подключения.

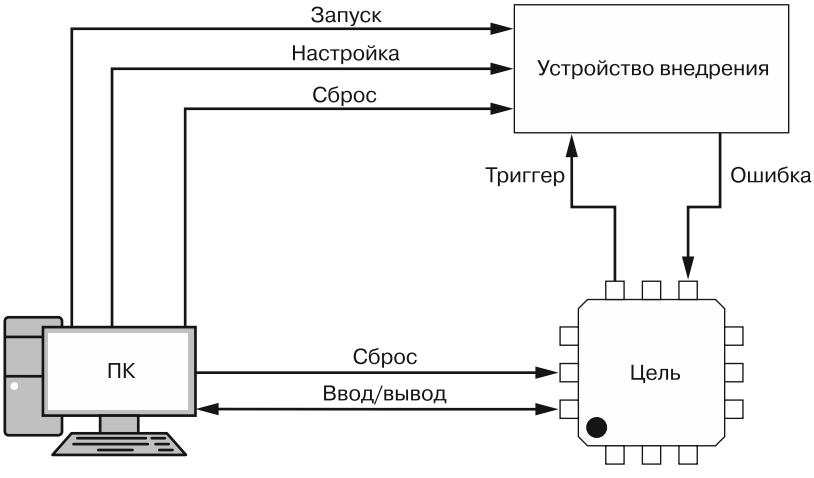

Рис. 4.3. Соединение между ПК, устройством внедрения и целью

Показанное на рис. 4.3 устройство внедрения — физический прибор, который выполняет внедрение ошибок. На данный момент мы просто предполагаем, что он может каким-то образом внедрить ошибку в целевое устройство одним из методов, которые мы кратко описали (тактовый сигнал, напряжение и т. д.). Целевое устройство должно само запустить устройство внедрения ошибки с помощью сигнала триггера, чтобы синхронизироваться с ним. Этот триггер обычно отправляется непосредственно в устройство внедрения, так как это позволяет добиться гораздо большей точности синхронизации, чем с участием ПК. ПК будет управлять связями с целевым устройством, поскольку нам нужно записывать множество выходных данных с устройства. Поскольку время — очень важный фактор, узнать больше об общей настройке мы можем, рассмотрев взаимодействие.

На рис. 4.4 показана общая диаграмма последовательности, описывающая взаимодействие между ПК (контролирующим все устройства), устройством внедрения и целью. Вы можете считать устройство внедрения подключенным к ПК через стандартный интерфейс наподобие USB.

Эта временная диаграмма показывает, что сначала мы настраиваем устройство внедрения, задавая параметры, которые хотим протестировать. В данном примере у нас также задаются *задержка* и *длина сбоя*. После триггерного события от цели устройство внедрения ожидает в течение задержки сбоя, а затем внедряет

его на заданное время. После внедрения ошибки мы наблюдаем результат целевой операции.

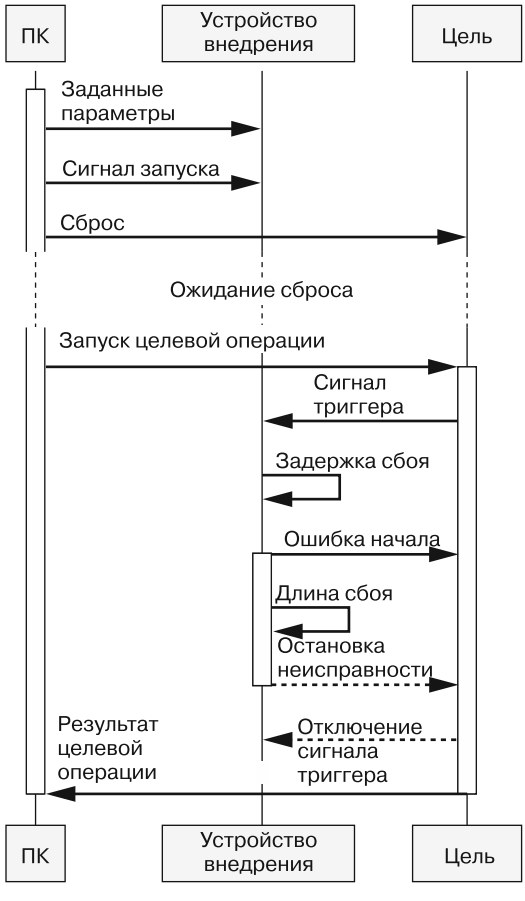

Рис. 4.4. Последовательность операций по внедрению единичной неисправности, инициированной ПК, который управляет и устройством внедрения, и целью

Отправка команды на целевое устройство

На целевом устройстве запускается процесс или операция, в которую вы хотите внедрить ошибку в соответствии со сценарием. В зависимости от операции, команда может быть отправлена по RS232, JTAG, USB, сети или другому каналу связи. Иногда для запуска целевой операции достаточно просто включить устройство. В предыдущем примере с OpenSSH вам нужно подключиться к демону SSH по сети, чтобы отправить пароль, который запускает операцию проверки пароля.

Получение результатов от целевого устройства

Далее нам требуется узнать, привела ли введенная ошибка к какому-либо интересному результату. Типичный способ получить результат — отслеживать связь с целевым устройством на наличие любых кодов результата, статусов или других сигналов, через которые можно осуществить внедрение. Старайтесь отслеживать и записывать всю информацию из канала связи на самом низком уровне.

Например, при последовательном соединении нужно отслеживать все байты, движущиеся по линии связи в обоих направлениях, даже если поверх этого выполняется более сложный протокол. Смысл в том, что устройство должно ошибаться. Выдаваемые данные могут быть необычными и не соответствовать обычному протоколу связи. Нужно сделать так, чтобы синтаксические анализаторы протоколов на вашей стороне не препятствовали перехвату ошибки от устройства. Перехватывайте все, а разберетесь позже. В случае с примером OpenSSH вы должны перехватывать весь сетевой трафик от цели, не доверяясь одному лишь журналу SSH-клиента.

Управление сбросом целевого устройства

Скорее всего, вы не раз сломаете целевое устройство, прежде чем ваши эксперименты увенчаются успехом, поскольку каждый из них может вызвать неопределенное поведение или состояние. Вам понадобится какой-то способ сбросить устройство в известное состояние. Один из способов активировать горячий сброс — отправить сигнал на соответствующий контакт или кнопку. Этого обычно достаточно, хотя иногда устройство не перезагружается должным образом. В таком случае вы можете выполнить холодный сброс, сбросив напряжение питания ядра или самого целевого устройства. Выполняя прерывание напряжения питания, необходимо отключать его ровно на время, достаточное для чистого сброса (сделав это слишком быстро, вы можете вызвать ненужную вам ошибку). Если это невозможно, то можно использовать дешевый удлинитель с USB-управлением, но он тоже может сломаться. Оба конца канала связи могут выйти из строя, если ваше устройство выдает странные данные. Хост должен будет снова распознать целевое USB-соединение, прежде чем вы сможете продолжить. Управляющий код на хосте должен предвидеть и пытаться решить любую из этих проблем. В примере с OpenSSH устройство, работающее на OpenSSH-сервере, после сброса должно автоматически перезагружать сервер.

Управление триггером

Триггеры — это электрические сигналы, генерируемые целевым устройством. Устройство внедрения использует их для синхронизации своей работы с определенными операциями в целевом устройстве. Использование стабильного триггера с минимальным дрожанием сигнала упрощает точный выбор момента для внедрения ошибки. Лучший способ сделать это — запрограммировать целевое

устройство на генерацию триггера на любом из внешних выводов микросхемы, например GPIO, последовательном порте, светодиоде и т. д. Непосредственно перед срабатыванием триггерный вывод подтягивается к высокому напряжению, а после срабатывания — стягивается к низкому. Когда устройство внедрения получит сигнал триггера, оно подождет заданное время, а затем внедрит ошибку. Таким образом, вы получите устойчивую точку отсчета времени по отношению к целевой операции и сможете пробовать внедрять ошибки, меняя величину задержки. На рис. 4.5 приведен обзор целевой операции, триггера и времени ошибки.

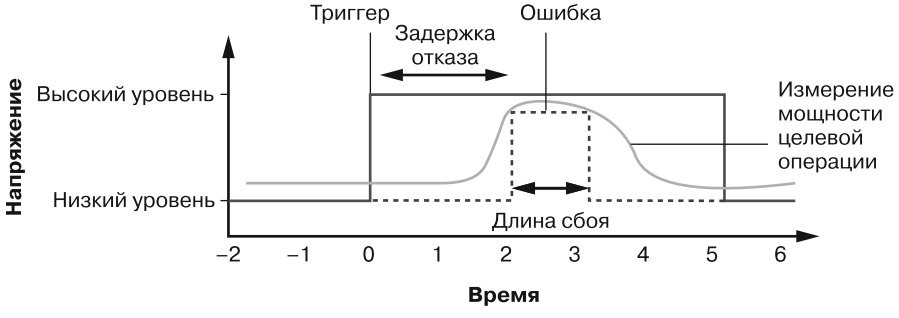

Рис. 4.5. Обзор целевой операции, триггера и времени отказа

Энергопотребление, измеряемое осциллографом, — целевая операция. Импульс, также измеренный осциллографом, — триггер, а сама ошибка представляет собой входной импульс, созданный устройством внедрения и заданный длительностью и амплитудой.

Несмотря на то что задержка после триггера *должна* быть постоянной, дрожание тактового сигнала на целевом устройстве может означать, что целевая операция не успевает выполниться за заданное время, а это значит, что вероятность успешного внедрения ошибки снижается.

Причины дрожания сигнала могут быть разными, поэтому в процессе описания вашего устройства нужно обязательно проверить, свойственно ли устройству непостоянство времени выполнения. Среди очевидных источников дрожания сигнала могут быть прерывания и большие фрагменты дополнительного кода между вашими инструкциями запуска и целевым кодом для внедрения. Но даже «простые» устройства (например, процессоры ARM Cortex-M) могут динамически оптимизировать машинные инструкции, и из-за этого задержка выполнения некой известной инструкции зависит от предыдущих выполненных инструкций *(контекста)*. Это означает, что при перемещении кода триггера в разные области может возникнуть неожиданно небольшая разница в количестве циклов. Многие устройства (в том числе ARM Cortex-M) поддерживают *барьер синхронизации инструкций* (instruction synchronization barrier, ISB), который вы можете добавить в код, чтобы «очистить» контекст перед выполнением кода триггера.

Если вам придется работать с устройством, не предоставляющим программный доступ для создания аппаратного триггера, то можно использовать программный триггер. Для этого нужно будет отправить команду для запуска операции с контролирующего хоста, отсчитать время задержки на контролирующем хосте, а затем запустить устройство внедрения, отправив ему программную команду. Чисто программное решение будет подвержено временным отклонениям, порождаемым всей программной системой управления. Внедрить ошибку все еще будет возможно, но надежно ее воспроизвести будет затруднительно.

В примере с OpenSSH вы можете перекомпилировать OpenSSH, добавив в код команду, которая генерирует триггер, а можете сделать программный триггер, отправив с управляющего хоста пароль на сервер OpenSSH, а затем команду запуска на устройство внедрения ошибок.

Мониторинг целевого устройства

Для отладки полученного стенда вам понадобятся средства мониторинга цели, каналов связи, триггеров и сброса. Здесь вам поможет логический анализатор или осциллограф. Выполните несколько целевых операций, не внедряя ошибок, а в процессе проследите за каналами связи, триггерами и линиям сброса. Все ли они работают нормально? При наблюдении за поведением цели также можно воспользоваться возможностями побочного канала (см. главы 8 и 9). У вас должна быть возможность видеть, например, длительность дрожания между сигналом триггера и выполняемыми операциями. Если вам кажется, что операции то спешат, то опаздывают, то виновато именно дрожание сигнала. Выполните несколько пробных внедрений и посмотрите, все ли работает должным образом.

Задачи мониторинга всегда подразумевают одну большую оговорку. В аналоговой области сам процесс измерения всегда влияет на цель. Нам не хотелось бы, чтобы осциллограф на линии VCC поглощал нужный нам скачок напряжения. Дополнительная нагрузка на провода всегда меняет форму внедренной ошибки. Если вам нужен постоянно подключенный осциллограф, то настройте его на высокое сопротивление и используйте щуп 10:1.

Прежде чем перейти к экспериментам с внедрением ошибок, трижды убедитесь, что все работает, а затем отключите все средства мониторинга, чтобы они не мешали результатам. Зачастую даже простые сбои при настройке, непредвиденная нестабильность или обновления операционной системы мешали провести хорошо продуманный эксперимент. А экспериментатор, потративший на это целые выходные, очень огорчался.

Выполнение модификаций, характерных для неисправности

Часто для успешного внедрения ошибки бывает необходимо физически изменить целевое устройство. В примере с OpenSSH для внедрения ошибки

в тактовый сигнал нужно было модифицировать печатную плату (PCB), чтобы найти точку введения тактового сигнала (другие возможности и тактики модификации мы обсудим ниже).

Чем тщательнее вы спланируете, запрограммируете и организуете все компоненты атаки, тем эффективнее будут эксперименты по внедрению ошибок. Тестовый стенд должен быть достаточно надежным, чтобы работать в течение нескольких недель и выдерживать любые возможные нештатные ситуации. Закон Мёрфи учит нас, что после миллиона успешных внедрений непременно что-то пойдет не по плану, но не обязательно в целевом объекте, а, может быть, в вашем стенде!

Методы поиска неисправностей

Теперь, когда целевое устройство подключено, а инструменты готовы, можно переходить к внедрению. Чего мы пока точно не знаем, так это когда, куда, сколько и как часто делать внедрения. Общий подход состоит в том, чтобы с помощью базового анализа цели, обратной связи от целевого устройства и капельки удачи найти выигрышную комбинацию параметров.

Для начала нам нужно определить, к какому типу ошибок чувствительна цель. В примере с OpenSSH мы сразу решили, что будем выполнять обход аутентификации, и предполагали, что знаем, куда внедрять ошибку и какого рода неисправности и параметры будут успешными. Возможно, мы могли вывести цель из цикла или повредить память. Для этого разрабатываются различные эксперименты и тестовые программы, которые помогут снизить чувствительность цели.

Далее мы рассмотрим пример внедрения ошибки в тактовый сигнал, подберем параметры и пройдемся по шагам, чтобы вы могли понять, как будет выглядеть эксперимент, когда вы соберете все воедино. Затем мы рассмотрим некоторые стратегии поиска, поскольку существуют различные методы обхода пространства поиска параметров больших ошибок.

Определение примитивов ошибок

Наличие программируемой цели позволяет вам экспериментировать и точно определять ее слабые стороны. Основная цель состоит в том, чтобы обнаружить примитивы ошибок и связанные значения параметров.

Примитив ошибки — это тип воздействия злоумышленника на целевое устройство при внедрении определенной ошибки. Это не сама ошибка, а вид достигаемого результата, например пропущенная инструкция или изменение определенных значений данных. Трудно точно предсказать, какие результаты могут быть