
Влад @lorc
Embedded разработчик
14,0 карма
0,5 рейтинг

Профиль | 1 Публикации | 469 Комментарии | 11 Избранное | 11 Подписчики

10 сентября 2016 в 14:35

Разработка → Немного о ARM Security Extensions (aka ARM TrustZone)

Системное программирование*, Программирование микроконтроллеров*, Информационная безопасность*

О чем эта статья

На Хабре уже несколько раз упоминали о SMM — режиме процессора x86/64 который имеет больше привилегий чем даже режим гипервизора. Нечто подобное есть и в процессорах архитектуры ARMv7 и ARMv8. Вычислительные ядра этих архитектур могут иметь опциональное расширение под названием ARM Security Extensions, которое позволит разделить исполняемый код, память и периферию на два домена — доверенный и недоверенный. Официальное маркетинговое название этой технологии — ARM TrustZone. Но технари чаще предпочитают говорить о security extensions.

Это будет обзорная статья, поэтому я не буду вдаваться в глухие технические дебри. Тем не менее технические детали будут присутствовать. Первая часть статьи будет посвящена вопросу зачем это всё вообще нужно, а вторая — как это работает в общих чертах. Если общество заинтересуется — следующая статья будет содержать больше технических деталей. Кому интересно — добро пожаловать под кат.

Зачем вообще нужны ARM Security Extensions

Хотя в названии и есть слово "security", эти расширения нужны не только реализации функций связанных с безопасностью в привычном понимании этого слова. Когда мы говорим "безопасность" мы обычно думаем о шифровании, ограничении доступа, TPM, безопасных хранилищах и тому подобных вещах. Security extensions позволяют реализовать эти все функции, но так же они используются для таких вещей, как например запуск и остановка процессорных ядер, перезагрузка системы и т.д.

Не совсем security функции

Все мы знаем что есть такая штука как BIOS (а теперь и EFI). Их задачей является первичная инициализация системы, загрузка OS, предоставление OS информации о системе, помощь OS в управлении компьютером.

Исторически сложилось так, что на ARM-based платформах нет (точнее не было) аналога BIOS. Обычно в ROM-коде присутствовал простенький загрузчик который находил и загружал полноценный загрузчик. Часто таким полноценным загрузчиком выступает u-boot. U-boot в свою очередь инициировал минимум периферии (например запускал контроллер DRAM), находил и запускал ядро linux. Ядро linux в свою очередь могло полностью вытереть из памяти u-boot, потому что он больше не нужен. И всё, после загрузки ядро было полностью предоставлено себе. Прямо в него (или в сбоку, в специальную структуру под названием device tree) была зашита вся информация о системе где оно работает. Все операции по управлению системой (например разгон процессора) ядру приходилось выполнять самому.

Это было неудобно по ряду разных причин. Например, производителям не хочется отдавать разработчикам ядра контроль над всякими "нежными" модулями типа кешей и шины. Или, например, процедура перезагрузки чипа требует каких-то хитрых манипуляций которые неудобно производить из кода ядра.

Короче, потребность в чем-то вроде BIOS существует. Только в ARM-системах он называет Secure Monitor и работает благодаря Security Extensions. По моему опыту чаще всего он используется для запуска и остановки дополнительных ядер в многопроцессорной системе, управления шиной, переброски выполняющегося кода между мощными и слабыми ядрами в ахритектуре big.LITTLE, активации режима гипервизора и тому подобных деликатных вещей.

Security функции

Security extensions позволяют запустить сбоку ещё одну OS со своим ядром и пользовательскими приложениями. Её обычно называют Trusted OS. У этой OS будет своя защищенная память и доступ к защищенной же периферии, например крипто-акселераторам. Двумя примерами таких OS является гугловая Trusty и опенсурсная OP-TEE. Существует консорциум Global Platform который выработал общие требования к таким OS под названием Trusted Execution Environment.

Гугл, например, использует свою OS для безопасного хранения ключей (андроидный сервис keymaster). Motorola в своё время использовала проприетарную реализацию безопасной OS для DRM. Там была вообще фантастическая система: пользователь мог хранить и проигрывать DRM-видео, при этом даже андроидное ядро ни в какой момент не могло получить доступ к памяти где хранились разжатые (или хотя бы расшифрованные) видео-фреймы. Но при этом Андроид мог, например, рисовать элементы управления поверх такого видео.

В защищенную OS можно загружать пользовательские приложения (естественно подписанные). Это позволяет например гонять там банковское приложение для платежей через NFC или виртуальную SIM-карту. Правда, скажу честно, на практике я с таким пока не сталкивался.

Так же secure extensions позволяют организовать Secure Boot: корнем доверия является ROM-код, который проверяет подпись следующего загрузчика, загрузчик может проверить подпись ядра обратившись к Secure Monitor. Ядро в свою очередь тоже может проверять подписи исполняемого кода. Таким образом основные компоненты будут защищены от изменения.Технология спорная, но некоторые разработчики android-устройств её очень любят.

О том как это всё работает

Информация о технических деталях открыта. Можно скачать ARM Technical Reference Manual и прочитать всё самому. Кроме того в Сети хватает всяких красивых презентаций. Я не буду вдаваться в глубокие дебри, просто опишу основные идеи. Я буду использовать терминологию из ARMv8, потому что она более последовательна, в сравнении с терминами принятыми в ARMv7.

Режимы процессора

В этом разделе будет приведено немного технических деталей. Если системное программирование для вас абсолютно темная тема — этот раздел можно и пропустить.

Собственно, все наверное слышали про кольца защиты в x86. В ARM-ах есть точно такая же штука, только тут они называются Exception Levels (сокращенно EL). Их может быть от двух до четырех (или шести, смотря как считать). Это режимы работы процессора. Чем выше Exception Level, тем большими привилегиями обладает код исполняющийся в нем.

Все ядра ARMv7 или ARMv8 поддерживают минимум два из них: EL0 и EL1.

На EL0 работают пользовательские программы (например ваш браузер). У кода исполняющегося в EL0 нет никаких привилегий: он (обычно) не может работать с периферией, перенастраивать MMU, запрещать (и разрешать тоже) прерывания, обрабатывать исключительные ситуации. Но браузеру этого всего и не нужно. А если всё-таки нужно, то об этом надо попросить ядро OS.

На EL1 работают ядро и драйвера. У них, соответственно, есть возможность работать с периферией, программировать MMU и далее по списку. Еще лет 10 назад этого было бы вполне достаточно. Но технологии не стоят на месте.

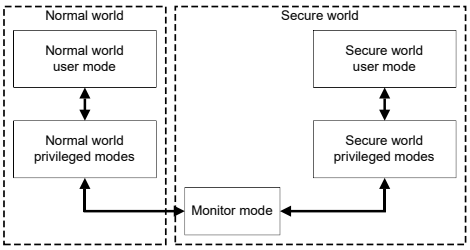
Ещё два EL могут появится если процессорное ядро включает дополнительные расширения: virtualization extensions и security extensions. При чем оба эти расширения опциональны (хотя и присутствуют во всех современных чипах) и процессор может иметь или любое из них, либо оба сразу. Немалая часть ARM Technical Reference Manual посвящена взаимодействию этих расширений.

Так, если если в ядре есть virtualization extensions, то появляется EL2. На этом уровне работает гипервизор. Так же MMU становится двухстадийным и появляется виртуальный контроллер прерываний. Но это совсем другая история которую нужно рассказывать отдельно.

Если в процессоре присутствуют security extensions, то появляется режим EL3 и режимы S-EL0 и S-EL1. Кроме того появляется понятие secure mode (и соответственно non-secure mode) — это режимы процессора ортогональные exception levels. EL3 обладает теми же привилегиями что и EL2, плюс ещё одной особенностью. Только в режиме EL3 код может переключить процессор между secure и non-secure mode. В чем же между ними разница? А разница только в значении одного бита — NS.

Дело в том, что security extensions — это не только расширения для вычислительного ядра. Эти расширения так же затрагивают MMU, контроллер прерываний и контроллер шины. Например, контролер шины проверяет значение этого самого бита NS при доступе к памяти и периферии. Если участок памяти помечен как secure, то доступ к нему можно получить только из secure mode. Это значит, что ни ядро обычной OS, ни гипервизор не смогут прочитать/изменить "безопасную" память. То же самое и с периферией. Если приходит прерывание которое помечено как secure, то это прерывание будет обрабатывать не обычная OS, а trusted OS из режима secure mode.

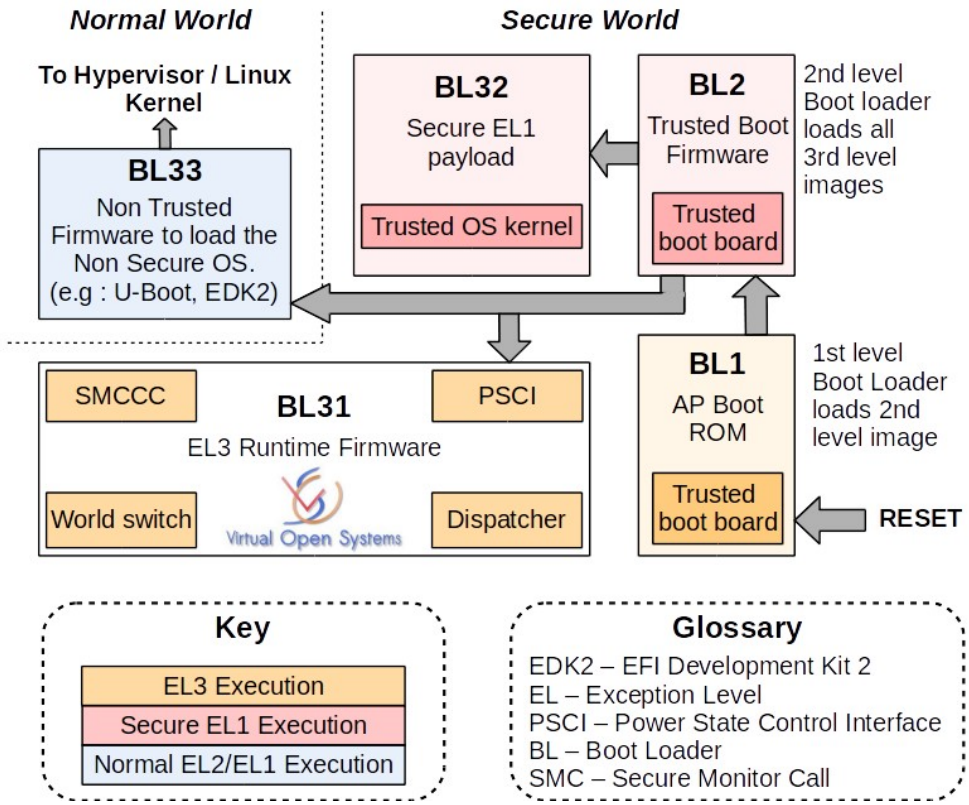
Получается, будто в одном процессоре живут два мира: normal world и secure world (это термины из официальной документации, если что). При чем secure world может вмешиваться в дела normal world, но не наоборот. В режиме EL3 работает secure monitor, который служит эдаким Хароном — позволяет попасть из одного мира в другой.



Как вы уже наверное догадались, режимы S-EL0 и S-EL1 — это аналоги EL0 и EL1 из normal world. В S-EL1 бегает ядро "безопасной" OS, а в S-EL0 — приложения (например та же виртуальная SIM-карта). На картинке выше используется терминология из ARMv7, но понять что к чему довольно легко (я надеюсь).

Взаимодействие Normal World и Secure World

Процессор всегда запускается в secure-mode (потому что иначе он туда никак не попадет). Загружается и инициализируется Trusted OS, после чего процессор переходит в non-secure mode и загружает обычную OS.



Казалось бы при всех своих возможностях secure world должен занимать доминирующее положение в системе. Но на самом деле всё наоборот. Большую часть времени он неактивен. Только когда обычной OS нужны какие-то услуги, она обращается к secure monitor и управление переходит в secure world. Таким образом именно normal world решает когда будет работать secure world. Это сделано для того, что бы не мешать пользователю работать с устройством, смотреть видео и слушать музыку. Ведь если secure world заберет управление в неподходящий момент, то это может нарушить работу normal world.

На самом деле у secure world есть возможность получить управление в обход normal world, используя прерывания. Например можно завести таймер который будет периодически вызывать secure world. Но так обычно не делают по причинам обозначенным выше. User experience превыше всего.

Для ядра linux есть набор патчей который позволяет обычным приложениям взаимодействовать с приложениями работающими в Trusted OS, согласно спецификациям GlobalPlatform TEE, о которых я уже упоминал выше. Таким разработчики могут писать приложения которые (с незначительными изменениями) смогут работать на любой совместимой Trusted OS.

Уголок параноика

Может ли secure world следить за вами? Теоретически — да. На практике я имел доступ к проприетарным trusted OS, которые затем устанавливались на пользовательские устройства. В их коде я не видел таких функций. Что, конечно же, ни о чем не говорит.

Если на вашем устройстве активирован secure boot, то у вас проблемы. В том смысле, что вы ничего не сможете сделать с кодом бегающим в secure world. Правда, вы так же ничего не сможете сделать и с кодом бегающим в режиме ядра.

К счастью, SoC'и с включенным secure boot довольно редки и их стараются не продавать всем подряд. Поэтому у вас скорее есть возможность заменить trusted OS на свою. Правда, ещё остается ROM-код с которым вы не сможете сделать почти ничего. И если secure monitor зашит в ROM-код, то опять же у вас проблемы. Но, есть SoC'и, на которых вы сможете устанавливать свой secure monitor. Например — Renesas RCAR H3. Так что ещё не всё потеряно.

Для желающих поэкспериментировать — есть возможность поднять Trusted OS на Raspberry PI. Как это сделать — написано в документации к OP-TEE.

ARM, SMM, security, security extensions, arm security extenions, SMC, TEE, OP-TEE, TrustZone

+26 5,8k 35

Влад @lorc карма 14,0 рейтинг 0,5
Embedded разработчик
Github Google Talk

Похожие публикации

+8 Бесплатный семинар «Oracle Cloud Security Day: Технологии на страже бизнеса», 2 февраля
1,2k 8 0

+17 Security Week 33: отключение Secure Boot, сортировка адресатов в GMail, последствия TCP-бага в Linux
8,9k 30 9

+72 Продолжение истории с UEFI Secure Boot
12,8k 16 346

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+29 По ту сторону баррикад вредоносного ПО. Исповедь малварщика
4,9k 55 16

+19 Эволюция CSS: от CSS, SASS, BEM и CSS–модулей до styled-components
2,7k 44 13

+15 А ваша служба является RESTful? Все что необходимо/обязательно знать про веб службы и REST
3,4k 70 40

+32 Пользователи Windows получили возможность работать с openSUSE (и Arch Linux)
16,3k 95 51

+8 ESP8266 + PCA9685 + LUA
812 14 1

Комментарии (21)

selenite 10 сентября 2016 в 16:00 # 0

> Secure Boot: корнем доверия является ROM-код, который проверяет подпись следующего загрузчика, загрузчик может проверить подпись ядра обратившись к Secure Monitor.

Есть ли какой-то тайный смысл загрузчику обращаться к Secure Monitor ради проверки подписи ядра? Если ключ прошит в загрузчик и не меняется, можно не обращаться ни за кодом, ни за ключами.

lorc 10 сентября 2016 в 16:47 0

Да, можно и так. Но обычно стараются сделать проверку подписей централизовано. Так, например, проще менять ключи
К тому же, если бы загрузчик сам проверял подпись, то в него нужно было бы встроить кучу криптографии. А так он делает ровно один secure monitor call, где передает указатель на буфер и размер. А монитор сам парсит контейнер с образом, находит нужный ключ и проверяет подпись. При чем монитор может воспользоваться аппаратным ускорителем криптографии.

Barafu 10 сентября 2016 в 16:49 0

Из Secure Monitor сложнее спереть ключ. У него своя память. Он может иметь(или не иметь) функцию «Загрузить дамп» и не иметь функции «скачать дамп». Хотя в итоге, конечно, всё равно сломают…

qw1 10 сентября 2016 в 19:00 *(комментарий был изменён)* 0

Зачем красть public key, тут криптография с открытым ключом.
Возможно, вы хотели сказать «подменить ключ», а не «спереть».
Но и в загрузчике подменить ключ не получится, изменённый загрузчик не верифицируется через Secure Monitor

d_olex 11 сентября 2016 в 04:09 +2

А доступ к защищенной памяти этого вашего EL3 можно получить с помощью DMA по аналогии с похожими атаками для IA-32, или контроллер памяти «знает» о security extensions и надлежащим образом защищает все что нужно?
Какой самый злой ARM девайс (в плане TrustZone, залоченного загрузчика, привязки к вендору, DRM, итд.) из тех что есть на масс маркете лучше взять что бы, так сказать, на реальном примере вкусить всю мощщу security extensions для задач безопасности?

d_olex 11 сентября 2016 в 04:14 +2

Можно ли «отравить» кэш инструкций для исполнения произвольного кода в EL3 по аналогии с похожей атакой для SMM?

lorc 11 сентября 2016 в 16:41 0

Насколько я знаю — нет. В армовской TRM'ке явно сказано что L1 instruction cache дропается при смене режима процессора.

Судя по описанию атаки, они генерируют запись в те физические регионы памяти, где хранится код SMM. Интеловская архитектура просто опускает такие операции. ARM же сгенерит Data Abort и на это всё закончится.

d_olex 12 сентября 2016 в 15:25 *(комментарий был изменён)* 0

> Судя по описанию атаки, они генерируют запись в те физические регионы памяти, где хранится код SMM.

Да, на интеле можно настроить MTRR так, что бы при обращении к каким-либо физическим адресам запись происходила непосредственно в кэш. Эта особенность применяется в частности для того, что бы использовать L2 кэш вместо RAM на ранних фазах platform initialization когда контроллер памяти еще не поднят.
Однако, на всех современных платформах фирмварь давно помечает SMRAM как non cacheable и лочит MTRR что бы код SMM нельзя было атаковать через кэш.

d_olex 11 сентября 2016 в 04:28 0

Еще такой вопрос: я правильно понимаю, что полностью открытых BSP для EDK2 которые включают в себя поддержку security extensions и trusted OS в природе нету?

lorc 11 сентября 2016 в 16:47 0

С EDK2 не работал, поэтому сказать не могу. Честно говоря даже не слышал о платформах где он используется.

Вообще на гитхабе ARM'а лежит совершенно открытый код secure monitor. OP-TEE тоже открыта. Там же на армовском гитхабе лежит и EDK2. При желании наверное можно собрать всё в кучу. Вообще, было бы круто, если бы все ARM-вендоры перешли на EFI, но пока предпосылок к этому я не вижу.

d_olex 11 сентября 2016 в 04:35 0

И вот еще вопрос: можно ли сгенерировать прерывание помеченное как secure или обратиться к EL3 каким-либо другим образом из кода который работает в EL0/EL1 под управлением аппаратного гипервизора, или это возможно только в EL2?

lorc 11 сентября 2016 в 16:53 0

Прерывание сгенерировать теоретически можно. Но надо что бы secure world настроил контроллер прерываний подходящим образом.
Правильный способ попасть в secure monitor — это smc.
Инструкция SMC (Secure Monitor Call) кидает в EL3 из EL1/EL2. Попытка вызвать её из EL0 приведет Undefined Instruction Exception.
Гипервизор может трапнуть SMC и таким образом виртуализировать secure monitor.

d_olex 12 сентября 2016 в 15:36 0

Другими словами, уязвимость в trusted OS можно использовать для выхода из-под гипервизора при наличии рута в гостевом домейне?

lorc 12 сентября 2016 в 16:43 0

Ну если вы сможете исполнять произвольный код в S-EL1 (или в EL3), то да, вся система у вас в руках. Так же как и в случае Intel SMM.

lorc 11 сентября 2016 в 16:35 0

Контроллер памяти висит на шине (так же как и контроллер DMA). Шина просто не даст ему работать с защищенными регионами.

Самые злые девайсы что я знаю — это SoCи от Texas Instruments. Но они забили на мобильные чипы и теперь ориентируются на automotive. Можно взять TI DRA7xxx. Вам нужна будет HS (High Security) версия, которую купить тяжеловато. Вообще, всё что связанно с безопасностью и криптографией очень тяжело добыть.

Можно взять тот же самый Renesas, но там всё куда проще. Насколько я знаю, secure boot там не построить.
А вообще на поиграться — хватит и RPI и даже Qemu. Ещё можно глянуть список поддерживаемых платформ в доках OP-TEE.
Функциональность (в этом плане) у них всех приблизительно одинакова.

d_olex 12 сентября 2016 в 16:34 0

Мне нужно что-то, где есть уже готовая реализация secure boot в том виде, в котором она должна существовать глазами OEM-ов, производителей SoC-ов и вендора платформы. Если сам код TrustZone будет наглухо проприетарный — не страшно, главное что бы спеки на чипы были доступны без NDA.
QEMU — не, я использую его для x86 и те фичи платформы которые он поддерживает всегда являются ограниченным подмножеством фич, которые поддерживаются реальными чипами доступными на рынке.

lorc 13 сентября 2016 в 14:56 0

Тут, к сожалению, сложно.
Я сталкивался с двумя реализациями secure boot которые реально используются в продакшене — одна от TI (точнее от TrustedLogic), вторая от Qualcomm. Но и у тех и у других всё так покрыто NDA, что даже получить документацию на обычный чип нелегко. А уж на secure-версию — так вообще.
При чем, дело не только в NDA, но ещё и в экспортных ограничениях на криптографию.

d_olex 13 сентября 2016 в 15:16 *(комментарий был изменён)* 0

С Qualcomm я немного знаком и имею доступ к части закрытой документации и исходников. Что-то он мне не очень нравится не только по причине его закрытости, но и по причине дичайшей навороченности: PMU, несколько ARM ядрер, несколько Hexagon ядер, куча периферии и все это работает под управлением нескольких ос запущенных одновременно (даже не считая trusted OS).
Надо будет посмотреть TI.

lorc 13 сентября 2016 в 15:31 0

Точно то же самое. Четыре здоровых ARM ядра, пара маленьких, которые управляют разной мультимедией (камера, ускорение видео), плюс DSP (который к счастью никто не использует), плюс отдельная прошивка для Audio Backend. Ну и GPU тоже со своей прошивкой, куда ж без него. И просто дичайше навороченная система power management. Хорошо, что на automotive чипах это не столь актуально.
Но всё равно мне TI как то милее квалкома. Они всё-таки повернуты лицом к комьюнити.

lorc 11 сентября 2016 в 16:57 0

Тут был вопрос от аккаунта read&comment по поводу обновления прошивки и образов tz, sbl1, sbl2.
К сожалению кнопка «отклонить» очень похожа на «ответить». И я случайно отклонил этот комментарий. Прошу прощения. Можете повторить его ещё раз, если хотите.

По сути вопроса — да, эти образы представляют собой цепочку загрузчиков. Но это не значит на 100% что активен secure boot. Загрузчики могут проверять, а могут и не проверять подпись следующего в цепочке. Всё зависит от производителя прошивки.

13_beta2 11 сентября 2016 в 22:22 0

Немного в обратной хронологии, но спасибо за ответ.

Сам вопрос для понимания был: «Наличие в обновлении прошивки образов tz, sbl1, sbl2 и т.д. говорит о наличии цепочки загрузчиков и как следствие использовании secure boot?»

Интересные публикации

Из жизни параллелилиста 2

Мониторинг приложений с помощью Pinba 1

ESP8266 + PCA9685 + LUA 1

Как тестируют ДГУ в дата-центре 0

Linux-2017: самые перспективные дистрибутивы 19