Виртуализация процессов в смартфонах с помощью TrustZone

С.Б. Калинин [1], А.В. Рейнюк [1], И.С. Теннант [1]

[1]Санкт-Петербругский государственный электротехнический университет «ЛЭТИ»,
Россия, 197022, г. Санкт-Петербург

Аннотация. В данной статье рассматривается использование технологии TrustZone для реализации виртуализации с использованием основных конфигураций гостевых операционных систем, включая системы с одним, двумя и несколькими гостями. Приводится обзор существующих решений, их архитектуры, производительность и ограничения. Особое внимание уделено применению гипервизоров для управления доступом к ресурсам и обеспечения информационной безопасности. Представленные исследования демонстрируют эффективность использования TrustZone в гетерогенных средах, включая встроенные и автомобильные системы, что подчеркивает перспективность данной технологии для систем с высокой степенью критичности.

Ключевые слова: TrustZone, гостевые операционные системы, виртуализация, информационная безопасность.

Abstract. This article examines the use of TrustZone technology for implementing virtualization with primary configurations of guest operating systems, including single, dual, and multiple guest systems. A review of existing solutions, their architectures, performance, and limitations are provided. Particular attention is given to the use of hypervisors for resource access management and ensuring information security. The presented studies demonstrate the effectiveness of TrustZone in heterogeneous environments, including embedded and automotive systems, highlighting the promise of this technology for high-criticality systems.

Keywords: TrustZone, guest operating systems, virtualization, information security.

Введение

Современные вычислительные системы сталкиваются с растущими требованиями к информационной безопасности и производительности, особенно в условиях использования виртуализации, которая обеспечивает сосуществование нескольких (гетерогенных) сред на одной вычислительной платформе. Долгое время виртуализация использовалась на настольных компьютерах и серверах для оптимизации использования и обеспечения максимальной доступности ресурсов. В настоящее время эта технология также широко распространяется в мобильных и встроенных устройствах [1], [2]. В данной статье рассматриваются основные подходы к реализации виртуализации на базе TrustZone, анализируются архитектурные особенности, производительность и ограничения, а также перспективы её применения в критически важных средах.

1. Описание рассматриваемой технологии виртуализации

Виртуализированная среда состоит из трех основных компонентов: аппаратной платформы, которая предоставляет аппаратные ресурсы для развертывания системы; гипервизора, также известного как диспетчер виртуальных машин (Virtual Machine Monitor – VMM), который виртуализирует оборудование; и одной или нескольких гостевых операционных систем или виртуальных машин (VM).

Технология TrustZone, хоть и реализована в целях защиты, позволяет использовать специализированную аппаратную форму виртуализации системы. Благодаря поддержке виртуального оборудования для выполнения в двух средах, а также другим функциям TrustZone, таким как сегментация памяти, появляется возможность обеспечить временную и пространственную изоляцию между средами выполнения. Незащищенное программное обеспечение работает внутри виртуальной машины, ресурсы которой полностью управляются и контролируются гипервизором, функционирующим в безопасном режиме. Виртуализация с помощью TrustZone не считается ни полной виртуализацией, ни паравиртуализацией, так как несмотря на то, что гостевые операционные системы (далее ОС) могут работать без изменений в незащищенном (недоверенном) режиме, они должны взаимодействовать с картой памяти и адресным пространством, которые они используют. Дальнейшее описание виртуализации будет рассмотрено для архитектур Armv7-A или Armv8-A.Опираясь на нынешний уровень развития технологий, можно выделить три типа системных конфигураций, поддерживаемых решениями виртуализации с помощью TrustZone [3], [4]: с одним, двумя и более гостями (рис. 1).

Рисунок 1 – Виртуализация с помощью TrustZone для ARM

Далее приведены существующие решения, связанные с конфигурацией с одним гостем (рис. 1(а)) [3], [5]. В рассматриваемой конфигурации гипервизор работает в режиме монитора, а гостевая ОС и её приложения функционируют в незащищенном режиме супервизора и пользовательском режиме. В разделе 1.2 описана реализация конфигурации с двумя гостями и представлена связанная с этим работа, [6]. В такой конфигурации гипервизор работает в режиме монитора, а защищенная гостевая ОС и её приложения функционируют в защищенном режиме супервизора и пользовательском режиме. Виртуальная машина, работающая в безопасном режиме, считается привилегированной, так как там не существует изоляции между режимами супервизора и монитора. В обычном режиме размещается (непривилегированная) виртуальная машина точно так же, как и в конфигурации с одним гостем. В конфигурации со многими гостями [4], [7] (раздел 1.3) немодифицированные гостевые ОС инкапсулируются между безопасным и обычным режимами: активная VM работает в незащищенном состоянии, а контекст неактивных VM сохраняется в защищенной области памяти. Такая настройка требует, чтобы гипервизор эффективно управлял общими аппаратными ресурсами: регистрами процессора, памятью, кэшем и MMU.

1.1. Один гость

Конфигурация с одним гостем представляет собой более простую системную архитектуру решения виртуализации с помощью TrustZone. Как показано на рис. 1(а), гостевая ОС работает под незащищенным периметром, а гипервизор работает в режиме монитора. Гипервизор имеет привилегированный вид на всю систему, а гостевая ОС имеет ограниченный доступ к системным ресурсам. Достоверная вычислительная база (trusted computing bases – TCB) системы ограничена кодом, работающим на стороне безопасного (доверенного) режима, то есть зависит от размера гипервизора. Память устройства и прерывания, назначенные гостевой ОС, настраиваются как незащищенные ресурсы и управляются непосредственно гостевой ОС, а остальные защищенные ресурсы находятся под строгим контролем гипервизора. Гостевая ОС управляет собственным MMU и строками кэша.

В литературе имеется небольшое количество решений, реализующих такую конфигурацию. Т. Френцель и соавторы [3] предлагают использовать TrustZone для реализации защищенной архитектуры Nizza [8]. Защищенный код состоит из небольшого гипервизора и набора непривилегированных компонентов, таких как защищенные драйверы устройств и защищенные службы. Обычный режим включает незащищенную гостевую ОС и ее приложения. Незащищенная гостевая ОС использует паравиртуализированные драйверы для отправки запросов на доступ к защищенным ресурсам, но также имеет драйверы для прямого доступа к устройствам, настроенным как незащищенные. Анализируемая система была применена и оценена на NVIDIA Tegra 2. Несмотря на использование многоядерной платформы, реализованное решение поддерживает только одноядерную конфигурацию. Авторы рассматриваемой публикации пришли к выводу, что использование возможностей виртуализации TrustZone привело к гораздо меньшему количеству необходимых изменений в Linux как незащищенной гостевой ОС по сравнению с паравиртуализацией, в то время как результирующие потери производительности варьируются от едва измеримых до 20 % в зависимости от характеристик рабочих нагрузок [3].

Х. Дуглас [5] описывает тонкий гипервизор, способный защитить FreeRTOS. Гипервизор с малым объемом памяти был задуман и разработан для обычных процессоров Arm, но то, как его можно реализовать и адаптировать для платформы с поддержкой TrustZone, все еще является предметом для обсуждения: гипервизор может быть изолирован в безопасном режиме, а FreeRTOS может работать в незащищенном периметре. Гостевая ОС может управлять собственной памятью и виртуальным адресным пространством, а также самостоятельно обрабатывать собственные исключения. Гипервизор отвечает за загрузку системы, а также за правильное назначение аппаратных ресурсов обоим режимам.

1.2. Два гостя

Система с двумя гостевыми ОС из-за точного соответствия количества консолидированных виртуальных машин и количества виртуальных состояний, поддерживаемых процессором, является наиболее часто используемой конфигурацией существующих решений виртуализации с помощью TrustZone. Как было проиллюстрировано на рис. 1(b), каждая гостевая ОС работает в своем независимом режиме, а гипервизор работает в режиме монитора. Данная конфигурация часто используется для систем со смешанной степенью критичности, где функции реального времени должны быть полностью изолированы от помех, не связанных с реальным временем. Как правило, ОС реального времени (Real-Time OS – RTOS) работает в безопасном режиме, а ОС общего назначения (General-Purpose OS – GPOS) в обычном режиме. Как только привилегированное программное обеспечение запускается в безопасном режиме, защищенная гостевая ОС получает полное представление обо всей системе, то есть она является частью TCB-системы. RTOS обычно имеют меньший объем памяти, что делает их перспективными для подобных конфигураций. Большинство существующих решений обычно реализуют асимметричный планировщик или планировщик бездействия, который требует, чтобы GPOS специально планировалось во время бездействия слотов RTOS. Память, устройства и прерывания обычно разбиваются один раз во время загрузки. GIC настроен для обработки защищенных прерываний как FIQ, а для незащищенных прерываний – как IRQ. В результате такого проектного решения защищенная гостевая ОС должна быть изменена на уровне ядра, в то время как незащищенная гостевая ОС работает без модификаций. При этом во время переключения между режимами не требуется выполнять операции управления кэшем и MMU.

М. Серейя и И. Бертолотти предложили асимметричный подход к виртуализации для систем реального времени, использующий TrustZone [9], [10]. Приведённое в исследовании решение поддерживает выполнение RTOS параллельно с GPOS. Система оценивалась на эмулируемой платформе с ARM1176JZF-S. Опираясь на получившиеся результаты. авторы подсчитали, что для одно миллисекундного тика гипервизора ожидаемые потери производительности GPOS ограничены 0,13 %, однако авторы не указывают, учитывают ли представленные результаты потери за доступ к памяти. Сравнивая расходы аналогичных решений, описанных в данном разделе, мы можем выдвинуть предположение, что это не так.

SafeG [11] из проекта TOPPERS состоит из решения с открытым исходным кодом, которое использует аппаратные расширения Trust Zone для одновременного запуска двух разных сред – GPOS и RTOS. Монитор SafeG, являющийся наиболее привилегированным

программным компонентом, работает в режиме монитора и отвечает за обработку переходов между RTOS и GPOS. На этапе инициализации SafeG настраивает ресурсы (память и устройства), назначенные RTOS как защищенные, а ресурсы, связанные с GPOS, как незащищенные. Устройства могут совместно использоваться с помощью механизма, называемого повторным разделением [12], [13]. SafeG настраивает защищенные устройства для генерации прерываний FIQ и незащищенные устройства для генерации прерываний IRQ. В первой версии SafeG реализован принцип планировщика бездействия, однако позднее он был расширен за счет встроенного планировщика [12]. Система поддерживает несколько плат, включая NXP i.MX6Q и Altera Cyclone V SoC. Исследования на плате PB1176JZF-S (оснащенной процессором ARM1176JZF с поддержкой TrustZone) демонстрируют время выполнения в худшем случае (WCET) 1,5 мкс и 1,7 мкс для переключения RTOS на GPOS и наоборот.

Безопасная автомобильная программная платформа (Secure Automotive Software Platform – SASP) [14] реализует упрощенный подход к виртуализации, использующий технологию TrustZone для обеспечения изоляции между системой управления и информационно-развлекательной системой (IVI) автомобиля. Проект является совместным предприятием Корейского университета и «Hyundai Motor Company». SASP использует функции TrustZone для одновременного запуска RTOS (например, AUTOSAR), для запуска управляющего программного обеспечения и GPOS с программным обеспечением IVI. Каждая гостевая ОС работает в каждом режиме в соответствии с ее критичностью. Уровень монитора, называемый V-Monitor, отвечает за управление гостями, распределение прерываний, управление общей памятью и посредничество в доступе к устройствам и связи. SASP поддерживает как одноядерные, так и многоядерные конфигурации. GIC распределяет прерывания по каждому режиму, используя FIQ и IRQ согласно классической модели. Устройства доступны только для безопасного режима, что влечёт за собой необходимость паравиртуализации GPOS. Система была применена и оценена на NVIDIA Tegra 3 в четырехъядерной конфигурации: одно ядро выделено для RTOS (AUTOSAR 2.0), а остальные ядра работают под управлением симметричной многопроцессорной (Symmetric Multiprocessing – SMP) версии Linux. Экспериментальные результаты показывают, что производительность GPOS снижается в пределах 1% при выполнении арифметических операций и в пределах 5% при выполнении системных вызовов.

LTZVisor [1], [15] из проекта TZVisor2 – это облегченный гипервизор с открытым исходным кодом, поддерживаемый TrustZone, предназначенный для консолидации систем со смешанной степенью критичности. С. Пинто и соавторы начали с предложения незавершенной работы [15], а затем представили и описали готовую версию гипервизора [1]. LTZVisor реализует классическую конфигурацию ОС с двумя гостями: в безопасном режиме размещаются RTOS и гипервизор, а в обычном – GPOS. Две гостевые ОС совместно используют один и тот же центральный процессор (далее ЦП), но принцип асимметричного проектирования требует, чтобы GPOS планировалась, когда RTOS простаивает, при этом RTOS может упредить выполнение GPOS. Память, устройства и прерывания настраиваются и назначаются соответствующим разделам во время инициализации системы и не используются совместно виртуальными машинами. LTZVisor поддерживает архитектуру Armv7-A, но в настоящее время также идут работы по расширению поддержки архитектур Armv8-A и Armv8-M. Объем памяти гипервизора составляет около 3 КБ.

Экспериментальные результаты (на Xilinx ZC702) демонстрируют, что RTOS не имеет никакого снижения производительности, а виртуализированная GPOS демонстрирует снижение производительности на 2% при частоте переключения гостевой системы в одну миллисекунду. LTZVisor-AMP [6] реализует поддержку контролируемой асимметричной многопроцессорной конфигурации: одно ядро работает в защищенном режиме и содержит защищенное программное обеспечение (LTZVisor и RTOS), а другое ядро работает в незащищенном режиме и содержит незащищенное программное обеспечение (GPOS). Исследования показывают, что многоядерное расширение решает проблему зависания, возникающую на одноядерных платформах, когда RTOS не передает контроль над ЦП, а также обеспечивает значительные преимущества в производительности, когда RTOS претерпевает большие нагрузки.

О. Шварц и соавторы [16] представили уникальный подход к виртуализации для разделения, который позволяет переключаться между виртуализированным и невиртуализированным режимами выполнения посредством программных перезагрузок. Работа отличается от большинства существующих решений виртуализации с помощью TrustZone, так как была разработана без учета конкретной архитектуры процессора или аппаратных расширений. Тем не менее, авторы продолжают проводить исследования того, как данная концепция может быть реализована с помощью технологии TrustZone. Загрузчик, гипервизор, виртуализированный (доверенный) гость и защищенные службы могут храниться в защищенной области памяти и выполняться только тогда, когда требуется перезагрузка части системы. Выполнение остального программного обеспечения происходит в обычном режиме. Мягкая перезагрузка будет реализована с помощью специальных гипервызовов, реализованных с помощью SMC.

VOSYSmonitor [17] – продукт с закрытым исходным кодом, разработанный и поддерживаемый Virtual Open Systems. VOSYSmonitor поддерживает одновременное выполнение двух ОС, таких как критически важная для защиты среды и некритическая. VOSYSmonitor отличается от связанной работы, поскольку он реализован для архитектуры Armv8-A и предоставляет возможность запуска гипервизора XEN или KVM [18] в обычном режиме. Гипервизор, работающий в обычном режиме, должен поддерживаться Arm VE. VOSYSmonitor изначально поддерживает выполнение двух гостевых систем, а планировщик реализует классическую политику простоя. В многоядерной конфигурации он разделяет ядро между обоими режимами, настраивает безопасный режим (RTOS) и обычный (GPOS) для обработки FIQ и IRQ. VOSYSmonitor был протестирован на плате Arm Juno и Renesas R-Car H3. Для систем под управлением FreeRTOS (системный такт 1 мс.) и Linux, развернутой на плате Arm Juno, тест показал снижение производительности примерно на 0,5 % для одноядерной конфигурации. При многоядерном подходе виртуализированная система превосходит автономный Linux, однако обоснований подобных результатов авторами не приводилось.

1.3. Много гостей

В течение нескольких лет отсутствие масштабируемости с точки зрения количества поддерживаемых гостей было основной причиной того, что часть исследователей воспринимали TrustZone как ограниченный и плохо управляемый механизм виртуализации. Путь к поддержке нескольких гостей – это новое начинание. SierraVisor [19] из проекта OpenVirtualization Project представил поддержку одновременного запуска нескольких

операционных систем на любом устройстве Arm11 или Cortex-A9 с поддержкой TrustZone. Однако SierraVisor не обеспечивает полной пространственной изоляции между гостями. Все незащищенные гостевые ОС используют одно и то же незащищенное адресное пространство, что может привести к тому, что незащищенная гостевая ОС может легко скомпрометировать и помешать правильному выполнению оставшихся гостевых ОС. С. Пинто и соавторы [4] стали первыми, кто опубликовал конфигурации, где много гостей. Изменяя состояние защищенной памяти, устройств и прерываний во время выполнения, а также тщательно управляя общими ресурсами, авторы продемонстрировали, как несколько экземпляров ОС могут сосуществовать, полностью изолированные друг от друга на платформах с поддержкой TrustZone.

Как было проиллюстрировано на рис. 1(c), гипервизор работает в режиме монитора, в то время как несколько гостевых ОС инкапсулированы между обычным и безопасным режимами: активная гостевая ОС работает в обычном режиме, а контекст неактивных гостей сохраняется в безопасный режим. Поскольку гостевые ОС могут работать только в обычном режиме, TCB системы ограничен размером гипервизора. Память (включая MMU и кэш), устройства и прерывания должны тщательно обрабатываться гипервизором. Изоляция памяти обеспечивается перенастройкой состояния безопасности сегментов памяти во время выполнения. Объем памяти активной гостевой ОС настроен как незащищенный, а оставшаяся память настроена как защищенная. На каждом гостевом коммутаторе TZASC перенастраивается, а MMU и кэш необходимо очищать. Для достижения изоляции на аппаратном уровне устройства, назначенные гостевым разделам, динамически настраиваются как незащищенные или защищенные в зависимости от их состояния (активное или неактивное) с помощью TZPC. Прерывания управляются в соответствии с аналогичной стратегией: прерывания защищенных устройств конфигурируются как FIQ, а незащищенных устройств – как IRQ. Защищенные прерывания перенаправляются на гипервизор, а незащищенные отправляются непосредственно активному гостю. Прерывания неактивных гостевых разделов мгновенно настраиваются как защищенные, и их обработка может следовать различным подходам [4].

RTZVisor [4] – монолитный гипервизор с поддержкой TrustZone, реализующий строгую аппаратную изоляцию между несколькими экземплярами ОС. Гостевые ОС мультиплексированы в обычном режиме. Одновременно может работать только одна активная гостевая ОС, при этом контекст неактивных экземпляров ОС сохраняется в защищенной области. Сильная пространственная изоляция памяти и устройств обеспечивается за счет использования TZASC и TZPC. Временная изоляция достигается за счет циклического планирования. Незащищенные интерфейсы MMU и кэша совместно используются несколькими разделами, что требует от гипервизора при изменении расписания нового раздела выполнения нескольких операций обслуживания. Исследования показывают, что расходы на виртуализацию составляют менее 2% при частоте переключения гостевой системы 10 мс. Когда скорость переключения гостевой системы снижается, расходы увеличиваются в геометрической прогрессии, достигая примерно 8% при скорости переключения гостевой системы в 1 мс, что обусловлено операциями, связанными с кэшем и MMU, которые необходимо выполнять во время гостевого переключения.

Дж. Мартинс и соавторы предложили μRTZVisor [7], который означает гипервизор реального времени с поддержкой микроядра TrustZone и представляет собой расширенную

версию RTZVisor [20] для архитектуры, подобной микроядру, и объектно-ориентированной реализации. μRTZVisor нацелен на безопасность при помощи защищенного процесса разработки и отличается от связанной работы тем, что, как решение на основе микроядра, способен запускать почти немодифицированные гостевые ОС, а как решение с поддержкой TrustZone, обеспечивает высокую степень функциональности в реальном времени. Гипервизор был дополнен стратегией планирования, основанной на временных областях, которые могут иметь разные приоритеты и планироваться в соответствии с упреждающей циклической схемой. Исследования демонстрируют снижение производительности в среднем около 2% при частоте переключения гостевой системы 10 мс.

2. Получение конфиденциальной информации посредством взлома приложений смартфона

В связи с переходом в информационное общество смартфоны стали частью нашей жизни, и для их использования были разработаны различные приложения, которые можно установить на устройство. Среди них мобильные приложения для обмена сообщениями, которые зарекомендовали себя как еще одна платформа для создания и укрепления социальных сетей за счёт не только предоставления простых услуг по отправке сообщений, но и передачи мультимедийного содержимого и личной информации, благодаря чему такие приложения уже насчитывают более миллиарда пользователей.

Несмотря на то, что многие используют одно или несколько приложений Social Network Service, их безопасности не уделяется достаточное внимание. Некоторые подобные приложения не предусматриваются какую-либо защиту, из-за чего подвержены различным атакам. В приложениях, разработчики которых реализовали механизм защиты, остаётся проблема того, что их средства защиты работают на одном уровне с приложением. Это означает, что любой метод обеспечения безопасности, основанный на уровне приложения, например, запутывание и предотвращение декомпиляции, можно победить с помощью атак переупаковки, которые манипулируют потоком управления приложения.

Как только мобильное устройство подвергается воздействию злоумышленника с помощью переупакованного приложения, он может извлечь локальные файлы данных – телефонную книгу, фотографии, видео и различные личные файлы в соответствии с разрешениями приложения. Среди таких файлов могут быть файлы учетных данных, содержащие идентификационную информацию пользователя. В большинстве случаев эти файлы зашифрованы и из них невозможно извлечь важную информацию о пользователе, однако злоумышленник может использовать сервисы приложения точно так же, как и жертва, и, используя извлеченные файлы учетных данных, пройти идентификацию личности, благодаря чему удастся получить доступ к ранее недоступным данным пользователя.

Обратный инжиниринг приложений для Android.

Большинство приложений под мобильные устройства с Android ОС разрабатываются на языке программирования Java или его ответвлений. Поскольку код Java предоставляет идентификаторы классов, функций и переменных, а промежуточный язык – байтовый код Dalvik – легко расшифровывается, установочный файл приложения можно преобразовать в исходный код в том виде, в котором он существовал на этапе разработки. Так важная внутренняя логика работы приложения может быть легко раскрыта.

Используемые для определения схемы управления приложением методы будут приведены далее.

Статический анализ кода.

Используя инструменты декомпиляции, например Baksmali и IDA, злоумышленник реконструирует код, после чего становится возможным аудит программного кода, и с помощью инструмента статического анализа потоков данных, такого как FlowDroid [21], можно легко обнаружить даже сложный поток. По мере увеличения размера пакета становится все труднее постулировать область, которую необходимо проанализировать, и следующие три метода обеспечивают решение этой проблемы.

Динамический анализ кода.

Используя Java Debug Wired Protocol, можно динамически анализировать код. Аналитик устанавливает в отладчике точку остановки в определенной области и проверяет идентификатор переменной этого шага или поток и прочее. Файлы библиотеки, такие как .so, присоединяются с помощью IDA и динамически анализируются.

Атака переупаковки Android.

Атака переупаковки означает изменение исходного приложения и распространение фальсифицированного для выполнения произвольного стороннего кода. Обратимая структура Android-приложения и особенности кода Java упрощают эту атаку. Используя Baksmali и дизассемблер вроде IDA, можно легко идентифицировать поток управления исходным кодом и внедрить в переупакованное приложение новый функционал, который будет выполняться незаметно для пользователя в фоновом режиме [22].

Рассматриваемая атака создает угрозу информационной безопасности пользователей мобильных устройств, однако стоит обратить внимание на то, что подделка подписи исходного приложения, которая создается с использованием криптографических алгоритмов с открытым ключом, является неосуществимой. Фальсифицированное переупакованное приложение будет иметь другую, собственную подпись, отличную от исходного приложения. Таким образом, успешный исход описанной атаки прежде всего основывается на халатном отношении пользователя к вопросам информационной безопасности при установке приложений.

Методология атаки.

Большинство мобильных приложений для обеспечения непрерывного обслуживания имеют собственные файлы учетных данных, которые содержат идентификационные данные пользователя и файлы данных в частном хранилище «/data/data/package_name». Поскольку большинство сервисов должны ссылаться на уникальные данные пользователя, они полностью зависят от этих файлов. Другими словами, файлы учетных данных – это не только идентификационные данные пользователя, но и книга истории, содержащая запись об использовании.

Несмотря на то, что файлы учетных данных чрезвычайно важны, как было указано ранее, существуют некоторые критические проблемы, которые не решены разработчиками приложений.

Первая проблема заключается в том, что приложения не выполняют аутентификацию файлов учетных данных, что означает, что приложение не проверяет, соответствует ли идентификационная информация, содержащаяся в файлах учетных данных, реальному устройству. По этой причине, как только эти файлы подвергаются атаке переупаковки, потере телефона или другим уязвимостям, теоретически появляется

возможность создать клон некоторых приложений, которые будут работать точно так же, с такими же учетными данными на другом устройстве.

SMS, ARS-аутентификация.

Для регистрации услуги выполняется SMS- или ARS-аутентификация для привязки учетной записи пользователя к конкретному устройству. После того, как сервер отправит несколько сообщений на устройство, пользователь вставляет код аутентификации и повторно отправляет его на сервер. Рассматриваемый подход является надежным методом аутентификации. Однако она выполняется только один раз для удобства пользователя. По этой причине для клонирования приложения нужны только файлы учетных данных после аутентификации устройства.

Аутентификация устройства.

Сравнив информацию, хранящуюся в файлах учетных данных, и информацию, извлеченную из устройства, можно эффективно определять, являются ли локальные файлы учетных данных поддельными или нет. Однако, так как процесс аутентификации устройства работает на уровне приложения, его можно обойти использованием атаки переупаковки.

Вторичная аутентификация.

Блокировка приложения для защиты личной информации, использует вторичную аутентификацию, такую как блокировка графическим шаблоном и блокировка паролем. Это надежный метод защиты, так как такая ключевая информация не содержится в файлах учетных данных. К сожалению, поскольку его правильная реализация усложняет работу и увеличивает нагрузку на сервер, большинство приложений сохраняет эту информацию на локальном устройстве. Это означает, что процесс аутентификации можно обойти, манипулируя файлами учетных данных.

Получив информацию, хранящуюся на локальном устройстве жертвы, появляется возможность создать клон и обойти процедуру предотвращения подделок переупаковкой, как показано на рис. 2.

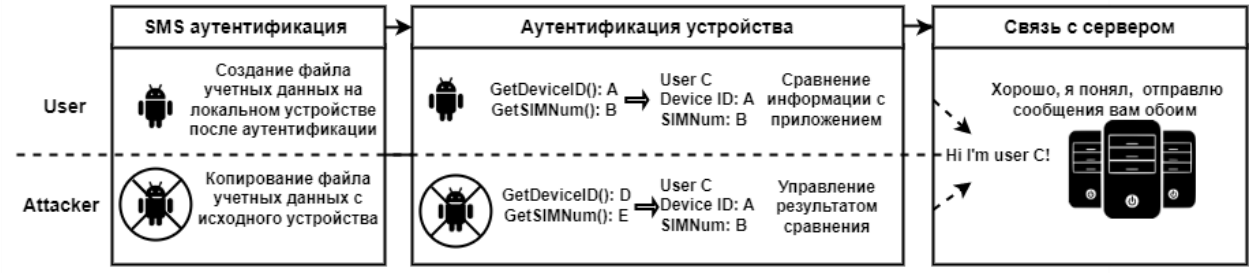

Рисунок 2 – Абстрактная модель атаки

Таким образом приложения, которые хранят конфиденциальные идентификационные данные локально, могут стать целью для атак. Это также означает, что несмотря на то, что информация, хранящаяся на устройствах, отличается, почти все приложения потенциально подвержены риску. Популярные приложения для здравоохранения хранят большое количество личной информации пользователей, в то время как большинство приложений для обмена сообщениями хранят содержимое разговоров. Кроме того, с появлением приложений, которые хранят информацию о кредитных картах, увеличивается не только универсальность приложения, но и его важность.

Некоторые приложения начали шифровать данные перед их сохранением, чтобы предотвратить утечку информации в случае атаки с переупаковкой или потери устройства. Однако какой бы сложный метод шифрования не использовался, данные должны быть расшифрованы, чтобы они отображались в виде обычного текста для пользователя при запуске приложения. Пользователю не нужно понимать, как расшифровывать данные.

Преимущество рассматриваемой атаки заключается в том, что тщательный анализ не требуется (рис. 3).

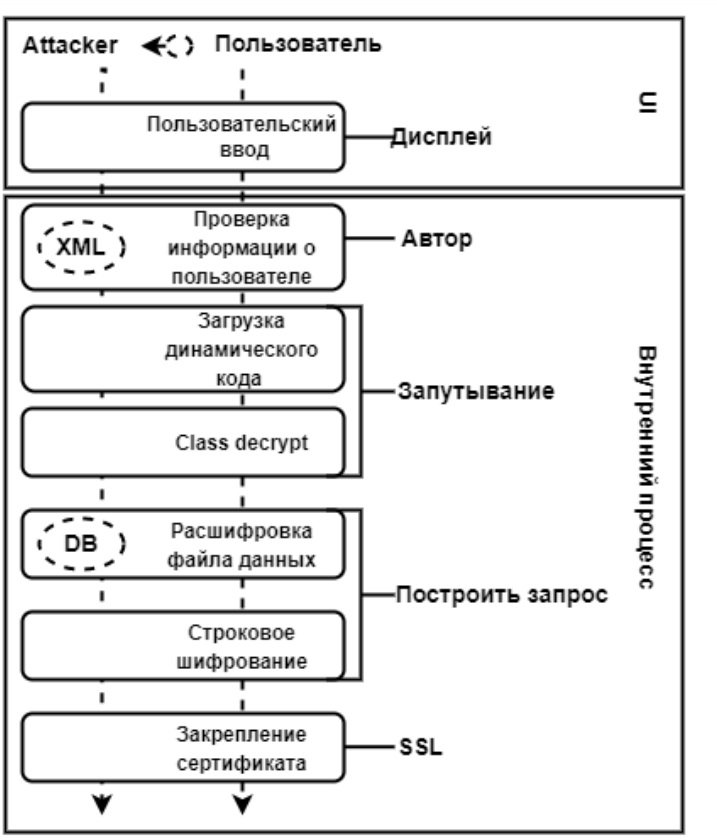

Рисунок 3 – Процедуры реального и дублированного приложений

Не имея информации о протоколах сетевого процесса или локальных методов шифрования и дешифрования данных, но правильно введя определенные данные, которые хранятся в файлах учетных данных, можно получить доступ на использование приложения. Хотя механизм безопасности различается в зависимости от приложения, как правило, существует потенциальный риск того, что, если это приложение для обмена сообщениями, можно просмотреть и отправить все сохраненные разговоры, а в случае приложения, использующего кредитную карту, можно напрямую использовать ее. для платежей.

Как уже отмечено ранее, текущие методы безопасности применяются на уровне приложения, и его методы проверки крайне ограничены. Например, приложение должно выполнять обнаружение несанкционированного доступа, обращаясь к самому установочному APK-файлу или к подписи через определенные вызовы функций (API), такие как getPackageInfo. Существует возможность без подробных знаний потока управления найти вызовы интересующих функций и изменить результат или аргумент на такой же, как и у исходного APK-файла.

Заключение

В данной статье рассмотрены важные аспекты безопасности мобильных приложений, с акцентом на угрозы, возникающие при взломе приложений для ОС Android, а также возможные решения, предоставляемые технологиями виртуализации, такими как TrustZone. Атаки на приложения, включая методы переупаковки и извлечение конфиденциальных данных, продолжают оставаться значимыми проблемами для информационной безопасности пользователей мобильных устройств. Однако использование технологий, обеспечивающих изоляцию и безопасность данных на аппаратном уровне, может существенно повысить защиту от таких угроз.

Особое внимание было уделено преимуществам применения TrustZone для создания защищенных сред, что позволяет эффективно управлять доступом к критически важной информации и уменьшать риски, связанные с несанкционированным доступом и утечками данных. Несмотря на высокую степень защиты, необходимы дополнительные усилия для обеспечения комплексной безопасности на всех уровнях, включая безопасность кода приложений, надежность процессов аутентификации и защиту локальных данных.

Таким образом, технологии виртуализации и аппаратного обеспечения, такие как TrustZone, являются перспективными инструментами в области информационной безопасности и могут существенно улучшить защиту ОС.

Благодарности

Работа выполнена в рамках государственного задания Министерства науки и высшего образования Российской Федерации № 075-00003-24-01 от 08.02.2024 (проект FSEE-2024-0003).

Список литературы

1. LTZVisor: TrustZone is the key / S. Pinto, J. Pereira, T. Gomes, A. Tavares, J. Cabral. // Leibniz International Proceedings in Informatics, Euromicro Conf. on Real-Time Systems, Vol. 76. Schloss Dagstuhl– Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany. pp. 4:1–4:22. URL: https://doi.org/10.4230/LIPIcs.ECRTS.2017.4;
2. A survey of mobile device virtualization: Taxonomy and state of the art / J. Shuja, A. Gani, K. Bilal, A. Khan, S. Madani, S. Khan, A. Zomaya // Comput. Surveys. Vol. 49. No. 1. Apr. 2016, pp. 1:1–1:36. URL: https://doi.org/10.1145/2897164;
3. ARM trustzone as a virtualization technique in embedded systems / T. Frenzel, A. Lackorzynski, A.Warg, H. Härtig // 12th Real-Time Linux Workshop, Nairobi, Kenya. 2010;
4. Towards a TrustZone-assisted hypervisor for real-time embedded systems / S. Pinto, J. Pereira, T. Gomes, M. Ekpanyapong, A. Tavares // IEEE Comput. Architect. Lett. Vol. 16. No. 2. July 2017. pp. 158–161. URL: https://doi.org/10.1109/LCA.2016.2617308;
5. Douglas H. Thin Hypervisor-Based Security Architectures for Embedded Platforms. Master’s thesis. Royal Institute of Technology. Skolan för informations- och kommunikationsteknik, Kungliga Tekniska högskolan, 2010;
6. Lightweight multicore virtualization architecture exploiting ARM TrustZone / S. Pinto, A. Oliveira, J. Pereira, J. Cabral, J. Monteiro, A. Tavares // Annual Conf. of the IEEE

Industrial Electronics Society. 2017. pp. 3562-3567. URL: https://doi.org/10.1109/IECON.2017.8216603;

7. µRTZVisor: A secure and safe real-time hypervisor / J. Martins, J. Alves, J. Cabral, A. Tavares, S. Pinto // Electronics. 2017. Vol. 6. No. 4. URL: https://doi.org/10.3390/electronics6040093;
8. The nizza secure system architecture / H. Hartig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, M. Peter. // International Conf. on Collaborative Computing: Networking, Applications and Worksharing. 2005. Vol. 10. URL: https://doi.org/10.1109/COL COM.2005.1651218;
9. Cereia M., Bertolotti I. C. Asymmetric virtualisation for real-time systems // IEEE International Symp. on Industrial Electronics. 2008. pp. 1680–1685. URL: https://doi.org/10.11 09/ISIE.2008.4677005;
10. Cereia M., Bertolotti I. C. Virtual machines for distributed real-time systems // Comput. Stand. Interfaces Vol. 31. No. 1. Jan. 2009. pp. 30–39. URL: https://doi.org/10.1016/j.csi.2007.10.010;
11. Sangorrín D., Honda S., Takada H. Dual operating system architecture for real-time embedded systems // Int. Workshop on Operating Systems Platforms for Embedded Real-Time Applications. 2010. pp. 6–15. Brussels;
12. Sangorrín D. Advanced Integration Techniques for Highly Reliable Dual-OS Embedded Systems. Ph.D. Dissertation. Nagoya University. Graduate School of Information Science. 2012;
13. Sangorrín D., Honda S., Takada H. Reliable Device Sharing Mechanisms for Dual-OS Embedded Trusted Computing // Int. Conference on Trust and Trustworthy Computing. 2012. pp. 74–91. URL: https://doi.org/10.1007/978-3-642-30921-2_5;
14. Secure device access for automotive software / S. W. Kim, C. Lee, M. Jeon, H. Y. Kwon, H. W. Lee, C. Yoo // Int. Conf. on Connected Vehicles and Expo. 2013. pp. 177–181. URL: https://doi.org/10.1109/ICCVE.2013.6799789;
15. Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone / S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, A. Tavares // IEEE Conf. on Emerging Technology and Factory Automation. 2014. pp. 1–4. URL: https://doi.org/10.1109/ETFA.2014.7005255;
16. Schwarz O., Gehrmann C., Do V. Affordable Separation on Embedded Platforms // Springer International Publishing, Cham. 2014. pp. 37–54. URL: https://doi.org/10.1007/978-3-319-08593-7_3;
17. VOSYSmonitor, a low latency monitor layer for mixed-criticality systems on ARMv8-A / P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, D. Raho // Leibniz International Proceedings in Informatics, Euromicro Conf. on Real-Time Systems. 2017. Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 6:1–6:18. DOI: https://doi.org/10.4230/LIPIcs.ECRTS.2017.6;
18. Dall C., Nieh J. KVM/ARM: The design and implementation of the linux ARM hypervisor // SIGPLAN Notes. Vol. 49, No. 4. 2014. pp. 333–348. URL: https://doi.org/10.1145/2644865.2541946;
19. SierraTEE. URL: https://www.sierraware.com/open-source-ARM-TrustZone.html;
20. Pinto S., Tavares A., Montenegro S. Space and time partitioning with hardware support for space applications // Data Systems in Aerospace Conf., European Space Agency, ESA SP,

Vol. 736. 2016. URL: https://www.researchgate.net/profile/Sandro-Pinto/publication/311847443_Space_and_time_partitioning_with_hardware_support_for_space_applications/links/585d44b608ae6eb8719ff501/Space-and-time-partitioning-with-hardware-support-for-space-applications.pdf;

21. S. Arzt, R. Siegfried, F. Christian, B. Eric, B. Alexandre, K. Jacques, L. T. T. Yves, O. Damien, M. Patrick. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps // 35th ACMSIGPLAN Conf. on Program. Language Design Implementation. ACM SIGPLAN Notices. Vol. 49. Issue 6. June 2014. pp. 259–269. URL: https://doi.org/10.1145/2666356.2594299;
22. Arxan Technologies. State of security in the app economy: Mobile apps under attack. URL: https://www.arxan.com/assets/1/7/State_of_Security_in_the_App_Economy_Report_Vol._2.pdf;
23. Android Open Source Project. URL: https://source.android.com/docs/security/features/encryption.