ПОРТФЕЛЬ РЕДАКЦИИ

БИТ

С. С. Агафьин, А. А. Краснопевцев

АТАКИ НА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВНЕШНИХ АППАРАТНЫХ КЛЮЧЕВЫХ НОСИТЕЛЕЙ

В связи с тем, что внешние аппаратные модули являются повсеместно распространенным средством хранения и обработки конфиденциальной информации, они постоянно находятся в поле зрения злоумышленников. Многие существующие виды атак на программное обеспечение модулей полностью исключаются при корректной реализации аппаратной платформы и платформы JavaCard [1], однако, согласно множеству исследований, даже в модулях, для которых документально подтверждено полное соответствие спецификациям, не всегда корректно встроены средства безопасности, что делает проведение атак возможным.

Модификация байт-кода

Самым эффективным способом внесения недекларированных возможностей в программное обеспечение платформы JavaCard является модификация преобразованного CAP-файла таким образом, чтобы можно было использовать архитектурные уязвимости платформы [2].

Одной из наиболее простых атак данного класса является атака «смешивания типов». Возможность ее проведения обусловлена спецификой хранения объектов классов и массивов в памяти внешних аппаратных модулей. Так, например, массив является структурой вида {число элементов; <элемент1>; <элемент2>;...}. Объекты класса хранятся в памяти в виде последовательности значений своих полей: {<поле1>;<поле2>;...}.

Интерпретация областей памяти, а именно определение того, какой тип данных (массив или объект) хранится в памяти, происходит лишь на основе инструкции байт-кода. Компилятор и конвертер JavaCard на этапе семантического анализа строго следят за тем, чтобы не происходило обращение к области памяти через ссылку некорректного вида (доступ к памяти объекта путем присвоения его адреса ссылке на массив). Однако они не могут обеспечить защиты от модификации уже сконвертированного файла.

Если создать в памяти объект, первым полем которого будет некоторое целочисленное значение, а затем путем модификации промежуточного представления присвоить ссылку на этот объект массиву, то в силу описанной выше особенности хранения массивов целочисленное значение будет интерпретироваться как длина массива, что позволит вывести при обращении к нему все содержимое памяти внешнего аппаратного модуля, если целочисленное значение, хранимое в поле объекта, будет достаточно большим.

От данной атаки существует несколько способов защиты. Первый — это обязательная генерация электронной подписи сконвертированного файла сразу после его создания. В этом случае целостность файла обеспечивается криптографическими методами. Единственная среда разработки, в которой использование этого метода защиты реализовано в полной мере, является NetBeans. Средства разработки (Eclipse и IDEA) не предоставляют этой возможности. При разработке с их помощью процесс компиляции и конвертации окончательной версии программного обеспечения из исходных файлов должен осуществляться некой доверенной стороной (например, администратором безопасности), которая иными инструментальными средствами сгенерирует электронную подпись для CAP-файла.

Другим распространенным способом защиты является статическая проверка соответствия типов ссылок при операции присваивания в процессе загрузки приложения на внешний аппаратный модуль. Чаще всего это реализовано штатными средствами, предоставляемыми разработчиками аппаратных платформ модулей, с помощью которых производится загрузка CAP-файла. Примерами таких инструментов являются SafeNet AppLoader и Gemalto JCard Manager. Существует также отдельное средство проверки соответствия типов, входящее в пакет разработчика JavaCard SDK, под названием Oracle CAP Verifier. Его использование не является обязательным в процессе разработки, однако, если он будет применен администратором безопасности перед подписью байт-кода, это поможет избежать ошибок, которые не смог выявить компилятор.

Утечка конфиденциальной информации через разделяемые интерфейсы

Платформа JavaCard реализует механизм разделяемых интерфейсов, объекты которых могут служить для несанкционированной передачи конфиденциальной информации.

Одним из подходов к компрометации данных через разделяемые интерфейсы является повторное разделение информации. Он заключается в том, чтобы сохранить ссылку на объект разделяемого интерфейса в некотором поле, которое, в свою очередь, принадлежит другому объекту разделяемого интерфейса. Как правило, классические средства разграничения доступа, реализованные в платформе JavaCard, не позволяют корректно обработать данную ситуацию и дают полный доступ к первому объекту при обращении через поле второго.

Для обеспечения защиты от данной атаки необходимо строго определять модель и политику безопасности для загружаемых приложений, а также непрерывно отслеживать потоки информации, которой обмениваются приложения в процессе взаимодействия. Типичным средством, применяемым в реализациях платформы для исключения данной угрозы, является многоуровневая мандатная модель доступа. Исполнение политики безопасности может быть выполнено с помощью монитора обращений, являющегося частью операционной системы внешнего аппаратного модуля, который вызывается при каждом использовании ссылочного типа данных виртуальной машиной, либо статически для проверки потока информации в приложении. Первый метод обращения является крайне затратным по вычислительным и емкостным ресурсам, что делает его неприменимым в большинстве случаев.

Второй же метод реализован в рамках проекта PACAP компании Gemalto, который использует статические проверки потоков информации между объектами приложения в заданной конфигурации. На практике этот подход означает, что разработчик приложения определяет уровни безопасности для объектов своего приложения и передает их перечень разработчикам внешнего аппаратного модуля. Они, в свою очередь, проверяют соответствие этих уровней остальным приложениям и в случае совместимости устанавливают приложение на модуль.

Механизм разделяемых интерфейсов в платформе JavaCard реализован таким образом, что проведение атаки вида «смешивание типов» возможно даже без модификации байт-кода.

Разделяемые интерфейсы позволяют обеспечить взаимодействие между разными программами, загруженными на внешний аппаратный модуль (между их контекстами безопасности).

Ссылки на экземпляры разделяемых интерфейсов могут быть корректно использованы даже через механизм ограничения взаимодействия.

Для того чтобы провести данную атаку, необходимо разрешить двум приложениям взаимодействовать через разделяемый интерфейс, но определить его в каждом из сконвертированных CAP-файлов по-разному. Это возможно, так как приложения загружаются на внешний аппаратный модуль отдельно друг от друга и не могут совершить проверку.

Так как оба приложения расположены в различных контекстах безопасности, ни одно из них не может получить доступ к данным другого без его предварительного запроса. Следовательно, необходимо в коде приложения, которое будет пересылать данные, определить некорректный тип (например, byte[] вместо short[]). Поскольку такой вид атаки использует исключительно штатные средства платформы JavaCard, эффективного средства защиты от него нет.

Использование некорректной системы транзакций

Другим штатным механизмом JavaCard, использование которого может повлечь нарушение конфиденциальности информации, является механизм транзакций. Он заключается в том, что платформой гарантируется атомарность исполнения последовательности команд. При успешной работе выполняются все команды из выделенной последовательности, а при ошибке на любом из этапов внутреннее состояние модуля возвращается в предшествующее первой команде. Для использования данного механизма JavaCard API предоставляет три метода: beginTransaction(), commitTransaction() и abortTransaction().

Система транзакций является самым уязвимым местом платформы JavaCard [3]. Несмотря на то что спецификацией гарантируется, что в случае прерывания транзакции все модифицированные поля возвращают свое начальное состояние, лишь небольшое число производителей внешних аппаратных модулей выполняют операцию очистки измененной памяти.

Простейшим способом использования данной уязвимости является случай, когда в прерванной транзакции успела выделиться память под некоторый массив, ссылка на который была присвоена некоторой локальной переменной. Тогда, если будет выделена память под другой массив уже после прерывания транзакции, в некоторых некорректно реализованных платформах ссылка на новый массив совпадет со ссылкой, которая по ошибке сохранилась в локальной переменной.

Эта атака может привести к серьезному нарушению безопасности, так как будет невозможно отслеживание корректности соответствия областей памяти и ссылок, связанных с ними. Эффективного средства защиты от данной атаки нет, поскольку она осуществляется с использованием уязвимости низкого уровня.

Другим способом использования вероятных проблем, связанных с реализацией механизма транзакций, является попытка включения в блок транзакции метода arrayCopyNonAtomic(). Поведение виртуальной машины JavaCard в этом случае не определено спецификацией. С одной стороны, копирование элементов одного массива в другой этой функцией гарантирует атомарность на уровне отдельных элементов, но не целого объекта, с другой стороны, будучи включенным в блок транзакции, действие этой функции должно быть полностью сброшено до момента исполнения первой инструкции блока транзакций. Как показывают независимые эксперименты [3], разработчики внешних аппаратных модулей по-разному обрабатывают эту ситуацию. Это может привести к тому, что одно и то же приложение JavaCard, загруженное на разные модули, будет вести себя по-разному. И если разработчик приложения рассчитывал, что память при неудачном исполнении функции будет полностью затерта, а разработчики модуля это не реализовали, то в некоторой области может оказаться часть конфиденциальной информации, к которой можно получить доступ, определив на ее месте новый массив, как описано выше.

Использование некорректного управления доступом к памяти

В связи с тем, что внешние аппаратные модули являются устройствами, вычислительные и емкостные ресурсы которых крайне малы, распределение памяти организовано по упрощенному алгоритму.

При вызове команды выделения памяти в «куче», она выделяется строго последовательно. Так как механизм сборки мусора не реализуется в карте, то подобное решение не приведет к фрагментированию памяти. Однако некорректная реализация обработки обращения к памяти со стороны API может привести к тому, что подобный подход станет причиной появления серьезной уязвимости, которая, будучи намеренно внесенной, может повлечь нарушение конфиденциальности информации, а также является крайне сложной для обнаружения.

В большинстве случаев виртуальная машина JavaCard при попытке доступа к неинициализированной памяти, находящейся за границами рассматриваемого массива, генерирует исключение. Однако в некорректных реализациях в случае, когда обращение производится к области памяти, которая инициализирована значениями другого массива, исключение не генерируется, а предоставляется доступ к этим сведениям, которые никак не связаны по данным с рассматриваемым массивом.

В качестве неполного закрытия этой уязвимости можно предложить выделять память под массивы всегда большего размера, чем того требует функциональность. Однако данное решение неприменимо при намеренном внесении уязвимости, а также увеличивает емкостные затраты приложения на некоторое относительное значение, что может быть критичным для устройств, ограниченных по ресурсам.

СПИСОК ЛИТЕРАТУРЫ:

1. Java Card Classic Platform Specification 3.0.4. URL: http://www.oracle.com/technetwork/java/ (дата обращения: 01.12.2013).
2. *Mostowski W., Poll E.* Malicious Code on Java Card Smartcards: Attacks and Countermeasures // LNCS. 2008. Vol. 5189. P. 1–16.
3. *Wang D., Ma C.* On the security of two smart-card-based remote user authentication schemes for WSN // Cryptology ePrint Archive. Report 581. 2012. P. 1–12.