Ссылки:

forcedentry citizen lab - https://citizenlab.ca/2021/09/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/

ipwn citizen lab - https://citizenlab.ca/2020/12/the-great-ipwn-journalists-hacked-with-suspected-nso-group-imessage-zero-click-exploit/

https://www.wired.com/story/ios-security-imessage-safari/ https://citizenlab.ca/2018/09/hide-and-seek-tracking-nso-groups-pegasus-spyware-to-operations-in-45-countries/

forcedry https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html

https://googleprojectzero.blogspot.com/2022/03/forcedentry-sandbox-escape.html

https://googleprojectzero.blogspot.com/2020/04/fuzzing-imageio.html

https://googleprojectzero.blogspot.com/2021/01/a-look-at-imessage-in-ios-14.html

Состав:

Введение (2 минуты)

• Краткий обзор актуальности кибербезопасности в современном мире.

• Введение в тему эксплойта FORCEDENTRY, его значимость и влияние на область безопасности операционных систем.

Основные принципы защитных механизмов (4 минуты)

1. NX (No Execute): Объяснение принципа работы и его роли в предотвращении выполнения кода из неисполняемой памяти.

2. DEP (Data Execution Prevention): Как DEP защищает систему от атак, основанных на выполнении кода в областях данных.

3. ASLR (Address Space Layout Randomization): Роль ASLR в усложнении эксплуатации уязвимостей путём рандомизации расположения данных и кода в памяти.

Анализ уязвимостей и эксплойта FORCEDENTRY (5 минут)

1. Обзор уязвимостей: Анализ типов уязвимостей, которые эксплойт FORCEDENTRY использовал для атаки.

2. Методы эксплуатации: Как FORCEDENTRY обходит защитные механизмы, такие как NX, DEP, и ASLR.

3. Адресация уязвимостей на уровне архитектуры: Предложения по укреплению архитектурной безопасности операционных систем.

Значение песочницы для безопасности приложений (3 минуты)

1. Введение в песочницы: Объяснение концепции и основных функций песочниц.

2. BlastDoor Sandbox: Рассмотрение механизма работы и его роли в защите iMessage от вредоносного кода.

3. Побег из песочницы: Как эксплойты, подобные FORCEDENTRY, нарушают изолированную среду выполнения.

Ошибки при проектировании и их последствия (3 минуты)

1. Распространённые ошибки при проектировании: Обсуждение, как недочёты в проектировании могут создавать уязвимости.

2. Примеры из реальной жизни: Анализ ошибок, которые привели к успешной эксплуатации уязвимостей в случае FORCEDENTRY.

Заключение и рекомендации (3 минуты)

• Обобщение основных моментов доклада.

• Важность непрерывного обновления и улучшения защитных механизмов.

• Предложения по дальнейшим исследованиям и разработке в области кибербезопасности.

Введение (2 слайд)

Современный мир все больше и больше зависит от компьютеров. Компьютеры отвечают за нашу безопасность, здоровье, финансы. Но помимо огромных преимуществ, такое внедрение цифровых систем подвергает нас риску быть взломанными. Всегда найдутся злоумышленники, которые попытаются обойти, завладеть или даже уничтожить систему ради собственной выгоды или на заказ. Наиболее популярным типом уязвимостей остаются атаки, направленные на манипулирование памятью. (картинка с графиком процента атак)

Именно про такие атаки мы и поговорим на примере эксплойта FORCEDENTRY. FORCEDENTRY (так же известный как уязвимость CVE-2021-30860) это эксплойт, разработанный Израильской компанией NSO Group (которая занимается кибербезопасностью и разработкой различного, в том числе, шпионского ПО). Суть данного эксплойта заключается в том, что из-за архитектурных ошибок в обработчике сообщений приложения iMessage злоумышленники смогли создать свой “компьютер внутри компьютера” при отправке PDF файлов зашифрованных кодеком JBIG2 под видом .GIF и, выйдя за “безопасный периметр“ (т н Sandbox), установили шпионское ПО Pegasus (от той же NSO Group). Самой интересной данного взлома особенностью, на мой взгляд, является тот факт, что FORCEDENTRY – это 0-click эксплойт, то есть, от пользователя устройства не требуется никаких действий для того, чтобы эксплойт начал свою работу.

Защитные техники (3 слайд)

Но прежде, чем разговаривать про взлом, давайте разберем основные механизмы и подходы защиты, которые применяются при проектировании и разработке систем. \*перечисление техник\*, про сендбоксинг немного попозже

NX (4 слайд)

NX (или No Execute) это механизм безопасности, реализованный на уровне процессора. Он помечает определенные участки памяти на данные и инструкции процессора, тем самым, делая невозможным выполнение кода на непредназначенных для этого участках. Достигается при помощи пометки области памяти, как неисполняемой при помощи специального бита. Такой подход, в теории, защищает от буферного переполнения и выполнения произвольного кода.

В iMessage NX используется совместно с ASLR и сендбоксингом для усложнения эксплуатации различных уязвимостей. Именно этот метод должен остановить выполнение вредоносного кода, даже если злоумышленник сможет его внедрить в память.

Однако NX не способен предотвратить взлом из-за некорректной изоляции компонентов. Например, хакер сможет взломать один компонент, и благодаря этому получить контроль над другим в обход NX.

DEP (4 слайд)

DEP (Data Execution Prevention) – механизм защиты, направленный на предотвращение выполнения вредоносного кода из неиспользуемых областей памяти. Во многом схож с NX, но может быть реализован как программно, так и аппаратно. Он помечает определенные области памяти, как недоступные для выполнения кода.

В теории, DEP (как и остальные механизмы) должен ограничивать исполнение, кода, внедренного злоумышленниками. Однако, будучи некорректно сконфигурированным, не сможет предоставить нужной защиты. Например, злоумышленники могут найти процессы или компоненты, не защищенные с помощью DEP. Благодаря тому, что может быть реализован программно, является более высокоуровневым по сравнению с NX. Так же он более гибок, чем аппаратный NX.

ASLR (Address Space Layout Randomization) (5 слайд)

ASLR (Address Space Layout Randomization) — это механизм безопасности, который усложняет атаку на память путём случайного изменения адресов памяти системных приложений и данных, исполняемого кода, а также стека и кучи.

ASLR предназначен для защиты от атак, которые зависят от предсказуемости расположения кода и данных в памяти, таких как переполнение буфера и атаки внедрения кода. В случае невозможности предугадать расположение нужных данных, взлом может занять значительно большее время.

Основными ошибками при разработке ASLR являются:

• Недостаточная энтропия: если рандомизация не обеспечивает достаточно широкого диапазона адресов, атакующие могут использовать методы перебора/

• Ошибки в реализации приложений: недостаточная защищенность самого приложения (например утечки информации о памяти), могут помочь атакующим обойти ASLR, предоставляя данные, необходимые для определения расположения важных областей памяти.

* адреса

ROP и JOP (6 слайд)

ROP

Return oriented programming – техника эксплуатации уязвимостей, основанная на выполнении произвольного кода, созданного из готовых инструкций. Используется для обхода NX, DEP. Исполняемый код формируется из гаджетов – небольших фрагментов кода, выполняющих ограниченный набор операций. После выполнения кода каждого гаджета и перед переходом к следующему происходит возврат в начало.

JOP

Jump oriented programming – эволюция ROP. Вместо инструкций возврата (как в ROP) использует переходы (jump). Это дает возможность фрагментам кода взаимодействовать друг с другом. При использовании техники JOP цепочка гаджетов может быть значительно более гибкой и эффективной.

Общий обзор (7 слайд)

Теперь, когда мы поговорили про основные принципы защиты, давайте перейдем к уязвимости. Как уже было сказано, FORCEDENTRY – эксплойт для iOS 14, позволявший из-за некорректной обработки медиа файлом нарушить работу операционной системе. Уязвимость была закрыта 13 сентября 2021 года с выходом iOS 14.8. FORCEDENTRY не полагается на единичную уязвимость, а использует цепочку уязвимостей для достижения конечной цели — выполнения произвольного кода на устройстве жертвы и установки шпионского ПО. Эта цепочка начинается с уязвимости в механизме обработки изображений.

Впервые обнаружен командой CitizenLab. Это лаборатория на базе университета Торонто. Занимается исследованиями в области кибербезопасности и прав человека.

Эксплойт был разработан компанией NSO Group для удаленной установки ПО Pegasus. Pegasus – мощнейшая шпионская программа для iOS и Android. Она позволяет получить доступ к данным на устройстве и отслеживать огромное количество его параметров. Используется для борьбы с терроризмом и преступностью, а так же, как показала практика, для слежки за оппозицией и несогласными.

Инициация через медиа файл: Атака начинается с отправки специфически созданных файлов (PDF форматированный с помощью JBIG2 и при этом помеченный, как GIF), который обрабатывается библиотекой ImageIO iOS. Эта библиотека используется для предсказания нужного формата файла (вне зависимости от расширения) и последующей обработки (чтение, преобразование, создания превью и тд) его. Стоит отметить, что трюк с подменой формата данных дает возможность использовать более 20 видов кодеков для взлома.

Небольшое отступление (8 слайд)

Прежде чем пройти дальше, поговорим про кодировки. JBIG2 – это формат, который был создан для сканирования изображений в PDF, когда людям требовалось максимально сильно сжимать файлы из за ограничений интернета и объёма накопителей. По сути, использует 2 основах техники:

1. Благодаря тому, что многие языки используют сравнительно маленькие алфавиты, JBIG2 старается разделить отсканированную страницу на отдельные символы при помощи простых алгоритмов сравнения без использования оптического распознавания текста. Т е, JBIG2 не знает, что он сканирует. После этого кодировщик составляет “карту”, помечая, где какие символы должны стоять. При декомпрессии получается так, что у нас есть всего 1 образец каждой буквы или символа, из которых уже формируются файлы. Недостатком такого подхода является тот факт, что кодировщик может случайно перепутать похожие символы и тем самым изменить суть документа (например, может поменяться стоимость какого-либо изделия или реквизиты организации). Это послужило одной из причин того, что JBIG2 ушел на пенсию.

2. Так как качество отсканированных документов может страдать, был придуман еще один подход: кодировщик может хранить референсный символ (например, тот, который больше всего похож эталон, заложенный в память), а так же, для каждой буквы или символа в тексте, разницу между тем, что он отсканировал, и референсом. Во время декомпрессии, используя различные логические операторы (например XOR, AND, OR), он восстанавливает исходный символ, значительно улучшая качество изображения.

Кодировка JBIG2 состоит набора сегментов (которые нужны для расстановки символов, обозначения новой страницы и тд). В библиотеке, которая использовалась Apple, сегменты представлены классом JBIG2Segment и его подклассами (Bitmap, SymbolDict, и другими). Bitmap представляет собой простой набор символов, а SymbolDictionary обьединяет несколько Bitmap.

К сегментам можно обращаться по их номерам, либо хранить через специальный лист (Glist) указатели на конкретные сегменты.

Код (9 слайд)

Уязвимость эксплуатируется во время обработки документов, которые были отправлены на устройство жертвы. После получения PDF файлов телефоном начинает свою работу парсер изображений. Его суть – получить формат полученных данных, обработать и подготовить для дальнейшей работы с ними.

Обратим внимание на представленный код:  
Guint numSyms = 0; // (1)

  for (i = 0; i < nRefSegs; ++i) {

    if ((seg = findSegment(refSegs[i]))) {

      if (seg->getType() == jbig2SegSymbolDict) {

        numSyms += ((JBIG2SymbolDict \*)seg)->getSize();  // (2)

      } else if (seg->getType() == jbig2SegCodeTable) {

        codeTables->append(seg);

      }

    } else {

      error(errSyntaxError, getPos(),

            "Invalid segment reference in JBIG2 text region");

      delete codeTables;

      return;

    }

  }

...

  // get the symbol bitmaps

  syms = (JBIG2Bitmap \*\*)gmallocn(numSyms, sizeof(JBIG2Bitmap \*)); // (3)

  kk = 0;

  for (i = 0; i < nRefSegs; ++i) {

    if ((seg = findSegment(refSegs[i]))) {

      if (seg->getType() == jbig2SegSymbolDict) {

        symbolDict = (JBIG2SymbolDict \*)seg;

        for (k = 0; k < symbolDict->getSize(); ++k) {

          syms[kk++] = symbolDict->getBitmap(k); // (4)

        }

      }

    }

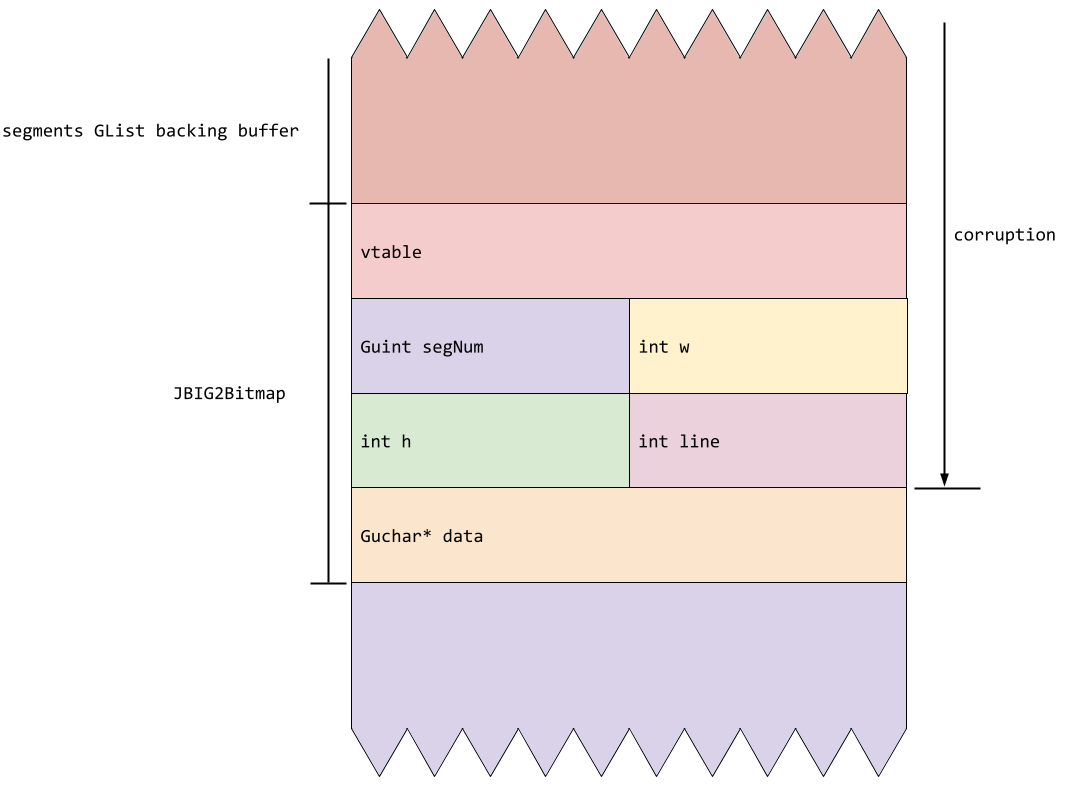
  }

Суть этого кода не очень важна. Нам нужно обратить внимание на переменную numSys (1) (32-х битное целое число). Благодаря заранее подготовленному PDF файлу с кодировкой JBIG2 и отсутствию достаточной валидации значений злоумышленники добились переполнения в (2) и получили небольшое значение numSys. Это, в свою очередь, вызвало создание буфера слишком маленького размера в (3). Далее идет перепить Bitmap в буфер в (4).

Такое развитие событий может привезти к записи огромного количества пустых данных, а как следствие, крашу парсера. Но логика обработки данных реализована так, что первые несколько записей мусорных данных повреждают резервный буфер Glist и перезаписывают указатели на Segment, заменяя их на Bitmap. Однако такой лавинообразный подход повредил бы все данные, не так ли? Нет. Bitmap наследует от Segment виртуальный вызов получения типа сегмента, однако Bitmap возвращает другой тип данных в этом вызове, что блокирует последующую запись и защищает злоумышленника от ненужного повреждения памяти.

Обработка BitMap (10 слайд)

Теперь, когда Glist поврежден, начинается обработка Bitmap, который отвечает за текущую страницу. JBIG2Bitmaps — это простые оболочки вокруг резервного буфера, хранящие ширину и высоту буфера (в битах), а также значение строки, которое определяет, сколько байтов сохраняется для каждой строки.

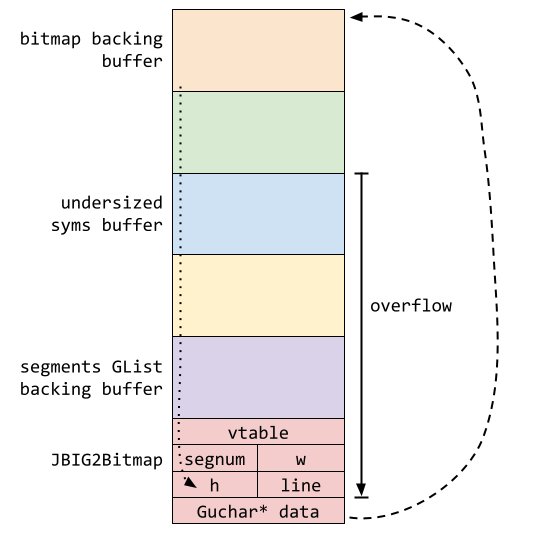


Далее, злоумышленник переписывает еще 3 указателя Bitmap и останавливает переполнение. К этому моменту информация, отвечающая за отрисовку текущей страницы, уже будет переписана. Благодаря тому, что iOS имеет прямой порядок байтов. Из-за особенностей структуры адресного пространства iOS, указатели скорее всего будут находится в памяти с адресами между 0x100000000 и 0x1ffffffff. А так как iOS имеет little-endian порядок байтов, то поля w и line будут перезаписаны с помощью 0x1 – наиболее значимой половиной указателя JBIG2Bitmap, а segNum и h получат наименее значимую половину казателя, скорее всего довольно случайное значение.

Из-за этого, значение h у текущей страницы JBIG2Bitmap будет неизвестным, но точно очень большим. А так как h используется для проверки границ выделенной памяти и должно отображать размер выделенного буфера, мы получим расширение границ канваса. А это, в свою очередь, приведет к тому, что следующие команды смогут быть прочитаны и записаны за пределами исходных границ буфера.

Ну как там с памятью? (11 слайд)

Подрезание кучи (heap groom) разместит буфер текущей страницы чуть ниже undersized буфера переменной syms, что приведет к тому, что если страницы JBIG2Bitmap неограниченна, она может записывать и читать свои собственные поля с помощью команд.



Записывая специально созданные 4 байтовый битмапы в координатах канваса, злоумышленник может записывать данные во все поля JBIG2Bitmap. А подбирая значения для w, h и line, он сможет записывать данные с произвольным оффсетом относительно буфера.

Теперь злоумышленник уже сможет переписывать данные по любым адресам, если он будет знать эти адреса. Все было бы легко, если бы смогли запустить произвольный JavaScript. Но мы все еще находимся в однопроходном парсере изображений. И тут наступает самая интересная часть.

А вот и логика (12 слайд)

Помните, что JBIG2 умеет работать с различными логическими операторами? А что, если использовать их не для работы с символами, но и для обработки отдельных битов? Это открывает огромные возможности, ведь злоумышленник может создать целые наборы команд. А так как буфер у нас не ограничен (еще один косяк с точки зрения проектирования), эти команды могут работать с фактически произвольной памятью. Объединяя логические операции в наборы команд, злоумышленники получают возможность создать целую архитектуру и запрограммировать ее, например, для побега из песочницы.

Это была та основа, за которой стоит FORCEDENTRY. Благодаря целому набору уязвимостей: неограниченным возможностям парсера, отсутствию защиты от переполнения (и соответствующей проверки чисел), злоумышленники смогли создать целую компьютерную архитектуру и запрограммировать ее нужным образом. Все эти ошибки по отдельности не кажутся такими уж и страшными, но, если их объединить, мы получаем возможность нанести серьезный вред безопасности. А это, как я считаю, является серьезным просчетом в архитектуре модулей iOS.

Методы эксплуатации (13 слайд)

Прежде чем перейти к побегу из песочницы давайте обсудим работу механизмов защиты.

Технические Детали Обхода NX (No Execute)

NX, как мы помним, защищает систему, запрещая выполнение кода из областей памяти, предназначенных только для данных. Однако, эта защита оказывается неэфективной, ведь FORCEDENTRY выбирает наборы инструкций (известные как гаджеты), которые уже есть в системе; создает цепочку гаджетов и выполняет нужный код для работы уязвимости, после чего перед ним открывается возможность работы с собственными наборами инструкций.

Технические Детали Обхода ASLR (Address Space Layout Randomization)

ASLR усложняет эксплуатацию уязвимостей путём рандомизации расположения ключевых областей памяти. Несмотря на это, FORCEDENTRY находит способы обхода ASLR и манипуляции с памятью: благодаря получению контроля над памятью из-за ошибок в разработке алгоритмов и библиотек, механизм ASLR оказывается неэффективным против этого эксплойта.

Подитог

Итак, благодаря проблемам архитектуры, а именно, отсутствию защиты от переполнения, проблемами с доступом к парсеру, а так же недостаточной изолированности злоумышленник получил практически неограниченный доступ к памяти и возможность создания собственной архитектуры

Песочница (14 слайд)

Теперь давайте поговорим про ничуть не менее важный механизм защиты

Концепция Песочниц

Песочница (или же Sandbox) — это защитный механизм для отдельного запуска программ, приложений или кода в строго контролируемой среде. Эта среда ограничивает доступ к ресурсам системы, таким как файлы, сетевые соединения, системные вызовы и интерфейсы операционной системы, предотвращая тем самым возможное вредоносное воздействие систему.

Технические Основы

• Сендбоксинг использует контейнеризацию для запуска программ в строго ограниченной среде.

• Песочница так же определяет, к каким ресурсам приложение может получить доступ. Это включает файловую систему, сетевые интерфейсы, системные вызовы и API операционной системы. Политики песочницы могут быть настроены для предоставления минимально необходимого уровня доступа.

• Песочница ограничивает права выполнения кода и доступа к файлам или библиотекам, применяя принцип наименьших привилегий. Это означает, что код выполняется с минимальным набором прав, необходимых для его работы.

Хоть сендбоксинг и может негативно сказываться на производительности устройства или приложений, это лучше, чем неконтролируемый доступ к системе со стороны различных программ.

Защита iMessage (15 слайд)

В iOS 14 Apple в очередной раз постаралась улучшить безопасность своей системы. Основными нововведениями с точки зрения системы и iMessage в частности стали: \*перечитать с слайда\*

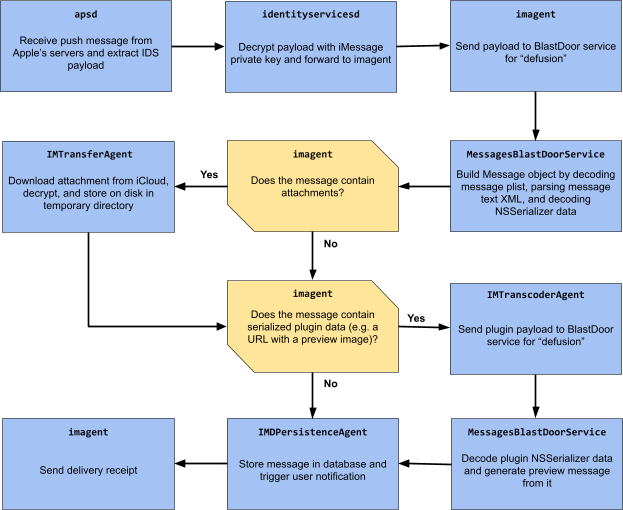
В новой версии iOS была улучшена ASLR: до обновления iOS 14 у Apple была серьезная архитектурная проблема, связанная с ним. Большинство библиотек, используемых системой, были доступны заранее в одной области памяти, причем адрес рандомизировался только один раз, при запуске системы. Теперь же, когда какой-то сервис пытается получить доступ к библиотекам, адрес этого участка памяти рандомизируется каждый раз после запроса.

Так же, для усложнения взломов, рассчитанных на перебор или работу с крашами приложений был введен троттлинг этих приложений. Теперь скорость работы каждого приложения или сервиса после его краша намеренно занижается, что значительно увеличивает время, требуемое на валом (вместо пары десятков секунд до нескольких десятков часов).

BlastDoor играет центральную роль в стратегии защиты iMessage, обеспечивая безопасную обработку входящих сообщений.

Обзор BlastDoor (16 слайд)

Этот механизм представляет собой изолированную среду, предназначенную для обработки и анализа данных полученных из сообщений, включая текст, изображения, видео и другие вложения, прежде чем они будут представлены пользователю. Цель BlastDoor — предотвратить выполнение вредоносного кода, эксплуатирующего уязвимости в системе обработки сообщений.



Побег из песочницы (17 слайд)

Теперь, когда мы знаем про нововведения в безопасности iOS 14, поговорим про то, как FORCEDENTRY их обошел. Итак, злоумышленник уже смог создать “компьютер” внутри песочницы. Следующая цель – это выйти за пределы изолированной зоны и установить уже настоящее шпионское ПО.

Для начала стоит поговорить о тех вещах, которые использовались для побега. В этом взломе хакеры использовали регулярные выражения и предикаты для достижения своих целей (NSExpression and NSPredicate). Благодаря тому, что поле isa (которое является указателем на класс, инстансом которого является текущий объект) у объектов в языке Objective C не было защищено кодами аутентификации указателей (PAC), архитектура, созданная в парсере, имела возможность использовать поддельные объекты Objective C, которые в свою очередь вызывали dealloc. Это приводило к тому, что выражение NSFunctionExpression десериализовывались и “инициализировались”.

Для связи с управляющим сервером будет использоваться модуль CommCenter, отвечающий за коммуникации. Он будет доступен из песочницы парсера через NSXPC.

Заметание следов (18 слайд)

Поговорим про заметание следов. Шпионское ПО на то и шпионское, чтобы остаться незамеченным. Функциональное выражение вызывает файловый менеджер системы, выбирает из временных файлов гифки, которые были использованны для взлома и удаляет их. Просто и элегантно.

NSXPC (19 слайд)

Теперь идем дальше. Основное ПО, Pegasus, находится на удаленном сервере. Для связи с ним можно воспользоваться модулем CommCenter. Для вызова методов из него в iOS используется такой механизм, как NSXPConnection (semitransparent rpc mechanism). Важно отметить, что для “пересылки” объектов через этот механизм необходимо, чтобы они имели свои протоколы, описывающие доступные для вызова методы. Для сериализации доступны подклассы любых классов без ограничения на наследственность. Это означает, что любой объект протокола, который объявляет определенный тип поля, так же принимает любой подкласс этого типа. Таким образом, протокол допускает использование любого подкласса (а это уже является критической угрозой безопасности).

Голос и PTRow (20 слайд)

Теперь можно передать массив состоящий из двух элементов: объекта синтеза голоса (странно, да?) и объекта PTSection с одним рядом PTRow. Но ведь для использования прототипа нам наверняка нужны соответствующие библиотеки. Для этого нам и нужен синтез голоса. Когда мы передаем объект синтеза голоса, рантайм Objective C вызывает метод initialize, что в свою очередь подгружает ряд библиотек для него. Одной из них и является библиотека для работы с PrototypeTool.

Что же тогда нас ждет в PTRow? Предикат. Десериализация и присвоение PTRow вызовет allowEvaluation. Таким образом, операционная система не будет выполнять никаких проверок предиката, что является, по сути, доступом к выполнению произвольного кода (еще одна критическая ошибка!).

Pegasus к нам приходит (21 слайд)

Такой же подход используется для “выполнения” еще несколько пейлодов в коммуникационном центре. Они делают следующее:

1. Создания URL по определенному адресу и сохранение его
2. Подгрузка дополнительных библиотек (так же, как с синтезом голоса)
3. Параметризация URL из первого шага с указанием точной модели и других параметров телефона
4. Создание полного URL адреса и получение данных сервера, и расшифровка

В этот момент исследователям не удалось проследить за дальнейшими действия эксплойта. Однако он уже имеет доступ сети и памяти устройства и ему ничего не мешает, чтобы скачать тот самый Pegasus. Таким образом, злоумышленник смог уже сбежать из сендбокса и запросить следующую информацию с управляющего сервера. В этот момент уже становится ясно, что хакеры нанесли серьезный ущерб с точки зрения безопасности, воспользовавшись большим количеством ошибок в архитектуре.

Заключение. Основные ошибки (22 слайд)

Итак, главной особенностью взлома, как мне показалось, является простота и элегантность использования дыр в ПО.

На примере эксплойта FORCEDENTRY мы познакомились с такими КРИТИЧЕСКИМИ ошибками при разработке, как:

Эксплойт использовал специально сформированные файлы, чтобы воспользоваться дырами в обработке медиа файлов внутри iMessage.

Отсутствие должной валидации входных данных привело к внедрению эксплойта, скрытого под маской PDF.

Эти файлы были разработаны так, чтобы вызвать ряд ошибок:

1. Переполнение целочисленной переменной
2. Получение контроля над памятью
3. Создание собственной архитектуры и побег из песочницы

FORCEDENTRY обходит защиты ASLR и DEP довольно изощренно: например, ASLR обходят с помощью утечек информации о расположении в памяти и получению контроля над ней. DEP же обходится таким образом, что инструкции выполняются в своей собственной архитектуре внутри парсера, независимо от основной памяти и процессора.

Из-за недостаточной изоляции злоумышленники смогли взаимодействовать из парсера и коммцентра с модулями, которые парсеру и центру даже в теории не нужны были. Это и доступ к специфичным библиотекам говорит о том, что модули были недостаточно изолированы, что привело к серьезным проблемам безопасности.

Заключение (22 слайд)

Данный эксплоит продемонстрировал, что ошибки, которые выглядят мелко и незначительно по отдельности, способны нанести серьезный ущерб и скомпрометировать такое сложное устройство, как смартфон.

Случай FORCEDENTRY подчеркивает важность комплексного подхода к безопасности программного обеспечения. Разрабатывая модули, нельзя забывать про то, как они смогут взаимодействовать друг с другом.

Взлом демонстрирует, что даже в высокозащищенных системах могут существовать сложные уязвимости, которые могут нанести серьезный ущерб безопасности.