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 это эксплойт, разработанный Израильской компанией NSO Group (которая занимается кибербезопасностью и разработкой различного, в том числе, шпионского ПО).

Так же мы затронем различные техники защиты системы от подобных атак.

Общий обзор (3 слайд)

Давайте поговорим про сам эксплойт. Как уже было сказано, FORCEDENTRY – эксплойт для iOS 14, позволявший из-за некорректной обработки медиа файлом нарушить работу операционной системе.

Суть данного эксплойта заключается в том, что из-за архитектурных ошибок в обработчике сообщений приложения iMessage злоумышленники смогли создать свою архитектуру внутри парсера при отправке PDF, выйдя за “безопасный периметр“ (т н Sandbox), установили шпионское ПО Pegasus. Самой интересной особенностью данного взлома, 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.

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

Инициация происходит через медиа файл: Атака начинается с отправки специфически созданных файлов (PDF форматированный с помощью JBIG2 и при этом помеченный, как GIF), который обрабатывается парсером медиа файлов iOS. Его задачами являются предсказание нужного формата файла и последующей обработки его.

Небольшое отступление (4 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. 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 из набора сегментов (которые нужны для расстановки символов, обозначения новой страницы и тд). В библиотеке, которая использовалась Apple, у сегментов имеются подклассы в виде Bitmap, который представляет собой набор символов и SymbolDictionary, объединяющий несколько битмап.

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

Код (5 слайд)

После получения сообщения устройством жертвы начинает свою работу парсер. Его цель, как уже было сказано ранее, обработать и сохранить полученный файл во временной директории.

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)

        }

      }

    }

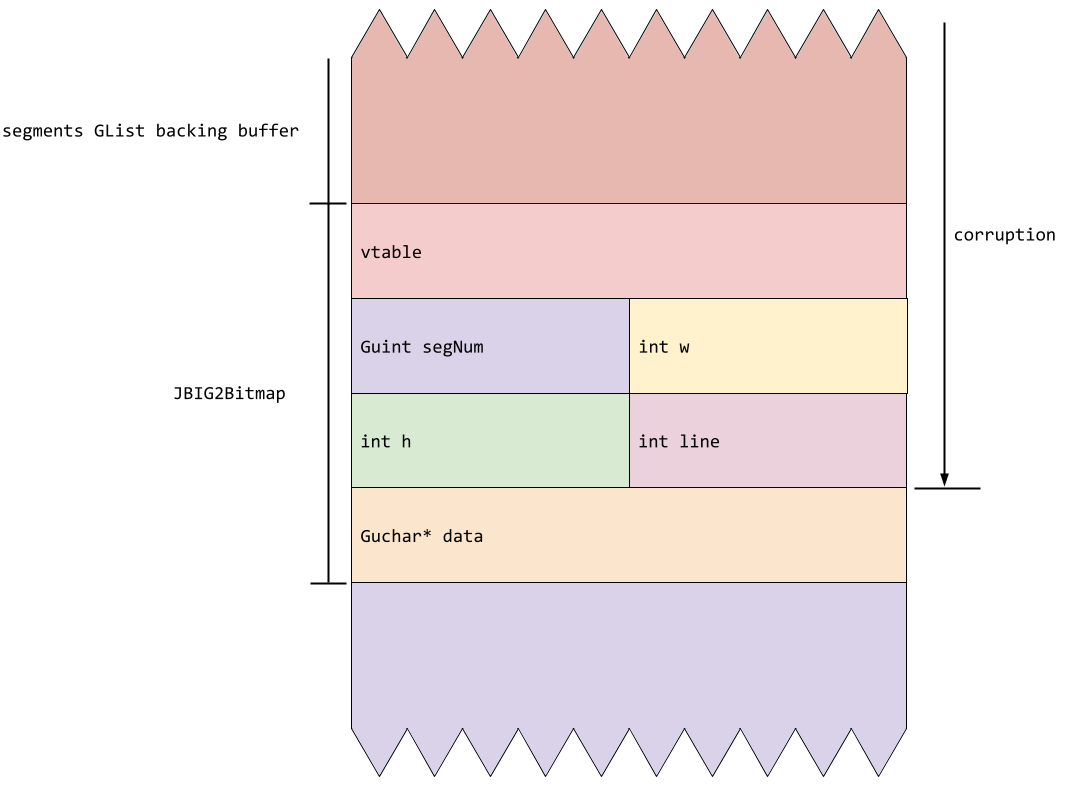
  }

Это фрагмент кода из того самого парсера. Его суть – получить размеры словарей и записать их в выделенную под них память. Переменная numSys (1) это 32-х битное целое число, которая хранит размер всех словарей. Благодаря заранее подготовленному PDF файлу и отсутствию достаточной валидации значений злоумышленники добились переполнения numSys при получении общего размера и получили небольшое значение в нем. Это, в свою очередь, вызвало создание буфера слишком маленького размера в syms. Теперь парсер пытается уместить все указатели на битмапы в выделенный буфер.

Такое развитие событий может привезти к записи огромного количества пустых данных (как раз из-за того, что буфер слишком маленького размера), 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 мог бы повредить бы все данные, если бы не одно но. Bitmap наследует от Segment виртуальный вызов получения типа сегмента, однако в отличии от сегмента, он возвращает другой тип данных в этом вызове, что блокирует последующую запись и защищает злоумышленника от ненужного повреждения памяти.

Обработка BitMap (6 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.

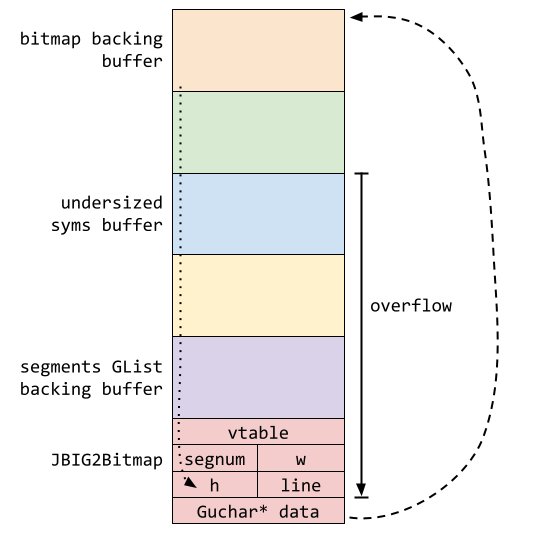


Далее, злоумышленник переписывает еще 3 указателя Bitmap и останавливает переполнение. К этому моменту информация, отвечающая за отрисовку текущей страницы, уже будет переписана. Благодаря тому, что iOS имеет прямой порядок байтов, поля w и line будут перезаписаны с помощью 0x1 – наиболее значимой половины указателя JBIG2Bitmap, а segNum и h получат наименее значимую половину казателя, какое то 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 “изображения”. 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? (7 слайд) \*заменить

Таким образом, буфер текущей страницы будет размещен чуть ниже undersized буфера переменной syms, что приведет к тому, что если страница JBIG2Bitmap неограниченна, она может записывать и читать свои собственные поля с помощью команд.



подбирая значения для w, h и line, злоумышленник сможет записывать данные с произвольным оффсетом относительно буфера.

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

А вот и логика (8 слайд) \*картинку?\*

Помните, что JBIG2 умеет работать с различными логическими операторами? А что, если использовать их не для работы с символами, а 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. Причем мы сможем реализовать любые логические операции из набора уже существующих. Например, логическое НЕ.

Это была та основа, за которой стоит FORCEDENTRY. Благодаря целому набору уязвимостей злоумышленники уже нанесли серьезный ущерб безопасности системы. Все эти ошибки по отдельности не кажутся такими уж и страшными, но, если их объединить, мы получаем вполне реальную угрозу.

Защитные техники (9 слайд)

Теперь когда мы поговорили про взлом, давайте обсудим несколько важных техник, которые используются для защиты устройств.

NX (10 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 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 (10 слайд)

DEP (Data Execution Prevention) – механизм защиты, направленный на предотвращение выполнения вредоносного кода из неиспользуемых областей памяти. Работает совместно с NX размечая области памяти на программной уровне. 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 конфигурируемым, чем NX. Так же он более гибок, чем аппаратная реализция, так как может использоваться в более старых системах.

ASLR (Address Space Layout Randomization) (11 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, получая данные, необходимые для определения расположения важных областей памяти.

* адреса

Методы эксплуатации (12 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 работает в контексте парсера, то есть находится в “безопасной памяти”.

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.

Песочница (13 слайд)

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. Она так же стремиться минимизировать привилегии, давай только те права, которые необходимы приложению, чтобы функционировать

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

Защита iMessage (14 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\*

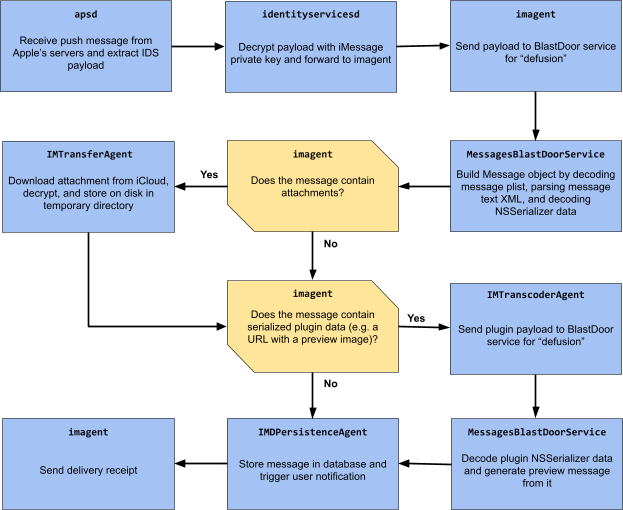
В новой версии iOS была улучшена ASLR: до обновления iOS 14 у Apple была серьезная архитектурная проблема, связанная с ним. Большинство библиотек, используемых системой, были доступны заранее в одной области памяти, причем адрес рандомизировался только один раз, при запуске системы. Теперь же, когда 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 (15 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 (16 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.

В этом взломе были использованы регулярные выражения и предикаты для достижения своих целей (NSExpression and NSPredicate). Благодаря тому, что поле, которое является указателем на класс, инстансом которого является текущий объект у объектов в языке Objective C не было защищено должным образом (кодами аутентификации указателей (PAC)), архитектура, созданная в парсере, имела возможность использовать поддельные объекты, которые в свою очередь вызывали десериализовывались и инициализировались.

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

NSXPC (17 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).

Заметание следов (18 слайд)

Поговорим про заметание следов. Шпионское ПО на то и шпионское, чтобы остаться незамеченным. Функциональное выражение, переданное в CommCenter, приведет к тому, что будет вызван файловый менеджер системы, который найдет gif файлы, хранящиеся во временном хранилище, и удалит их. Просто и элегантно.

Голос и PTRow (19 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 PrototypeTool нам наверняка нужны соответствующие библиотеки. Для этого нам и нужен синтез голоса. Когда мы передаем объект синтеза голоса, рантайм Objective C вызывает инициализацию, что в свою очередь подгружает ряд библиотек для него. Одной из них и является библиотека для работы с PrototypeTool.

Дальше нас ждет PTRow. Во время десериализации и присвоения система будет считать, что предикат является безопасным для нее, то есть не будет выполняться никаких проверко, что является, по сути, доступом к выполнению произвольного кода (еще одна критическая ошибка!).

Pegasus к нам приходит (20 слайд)

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

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

Дальнейшие действия эксплойта остаются неизвестными. 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 (21 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: \*слайд\*

Заключение (21 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.

Вопросы

1. Q: Каковы последствия использования эксплойта FORCEDENTRY и подобных для конечного пользователя?  
   A: Эксплойты, которые взламывают устройства без взаимодействия с пользователем позволяют значительно повлиять на работу устройства, внедрив Pegasus и подобные программы. Такие уязвимости подвергают опасности личные данные пользователей и ползволяют странам злоупотреблять технологиями. Например, Саудовская Аравия использовала Pegasus и Forcedentry для слежки за оппозицией.
2. Q: В чем ключевые отличия NX и DEP? (дополнительный вопрос, как работает DEP?)  
   A: NX реализован на уровне процессора и строго привязан к нему, что может привести к трудностям конфигурирования. Это делает его обход более сложным.  
   DEP же может работать как совместно с NX, так и отдельно, помечая области памяти самостоятельно.  
   DEP является более гибким, так как реализован программно. Это дает возможность ввести его в системы, на которых невозможно использование NX из за несовместимости аппаратного обеспечения.  
   (DEP работает размечая области памяти на неисполняемые программным образом)
3. Q: Как именно Apple исправили проблемы, позволявшие работать FORCEDENTRY?  
   A: Apple исправили переполнение целого числа а так же улучшили проверку входных данных в парсере. Так же, насколько я помню, была временна отключена библиотека, отвечавшая за JBIG2 и несколько других форматов. Так же были введены новые ограничения в правила сендбоксинга.
4. Q: Что делал Pegasus после получения доступа к устройству?  
   A: Пегасус – шпионское ПО, которое позволяет отслеживать огромное количество данных с устройства, оставаясь незамеченным. Работает на iOS и Android. Пользователи пегаса могут просматривать фотографии, входящие сообщения, прослушивать звонки, а так же следить за местоположением устройства.
5. Q: Какие принципы стоят в основе безопасной разработки ПО?  
   A: Принципы включают в себя максимальную изоляцию модулей и минимизацию привилегий. Нельзя так же забывать про валидацию ВСЕХ значений и использование защиты от переполнения и выхода за пределы допустимых границ. Важным аспектом так же считается шифрование всех данных, не только информации о пользователе.
6. Q: В чем заключаются уникальные отличия BlastDoor по сравнению с простым сендбоксингом?  
   A: В первую очередь, BlastDoor является конкретной технологией, а не концепцией защиты. Его особенность заключается в том, что в нем происходит вся предварительная обработка полученных сообщений. То есть он применяется не к приложению в целом, а к содержимому сообщений (файлы, ссылки, текст). Он так же именно анализирует информацию, а не просто изолирует среду.
7. Q: Какие еще типы эксплойтов популярны помимо 0-click?  
   A: Популярны так же: 1-click эксплойты (переход по ссылки, открытие вложения в почте; sql инъекции (внедрение sql команд); xss (cross site scripting – выполнение произвольных скриптов на клиенте); mitm (man in the middle, перехват и изменение трафика между двумя стронами, нпм телефон и роутер)