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

Современный мир все больше и больше зависит от компьютеров. Компьютеры управляют нашей безопасностью, здоровьем, финансами. Но всегда найдутся злоумышленники, которые попытаются взломать, обойти или уничтожить систему ради собственной выгоды или на заказ. Наиболее популярным типом уязвимостей остается повреждение памяти (картинка с графиком процента атак).

Именно про такие атаки мы и поговорим на примере эксплойта FORCEDENTRY. FORCEDENTRY (так же известный как CVE-2021-30860, это его индекс) это эксплоит, разработанный Израильской компанией NSO Group (которая занимается кибербезопасностью, делают очень мощное ПО, продают кому попало, даже возможно в РФ). Суть данного эксплоита заключается в том, что из-за некорректного кода в обработчике сообщений приложения iMessage злоумышленники смогли создать свой “компьютер внутри компьютера” при отправке pdf и JBIG2 файлов под видом .GIF и, выйдя за безлопастный периметр (т н BlastDoor Sandbox), установили шпионское ПО Pegasus (от той же NSO Group). Самой интересной особенностью, на мой взгляд, является тот факт, что от пользователя буквально не требуется никаких действий для того, чтобы его взломали.

Защитные техники

Но прежде, чем разговаривать про эксплойт, давайте разберем основные техники защиты от исполнения враждебного кода. Я бы хотел обратить особое внимание на такие методы, как NX, ASLR и DEP.

NX

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 с помощью специального бита (более низкоуровневый по сравнению с DEP). 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.

ASLR (Address Space Layout Randomization)

Принцип работы ASLR

ASLR (Address Space Layout Randomization) — это механизм безопасности, который усложняет атаку на память путём случайного изменения адресов памяти системных и приложений данных, исполняемого кода, а также стека и кучи. Он широко используется т.к дает возможность скрыть истинное расположение нужного кода или данных от злоумышленников.

ASLR предназначен для защиты от атак, которые зависят от предсказуемости расположения кода и данных в памяти, таких как переполнение буфера и атаки внедрения кода.

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.

DEP

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.

Общий обзор

Общий Обзор Уязвимостей

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.

Инициация через медиафайл: Атака начинается с отправки специфически созданных файлов (PDF, JBIG2 замаскированных под .GIF), который обрабатывается библиотекой ImageIO iOS. Эта библиотека используется для предсказания нужного формата файла (вне зависимости от расширения) и последующей обработки его. Стоит отметить, что трюк с подменой формата данных дает возможность использовать более 20 видов кодеков для взлома.

Небольшое отступление

Но что же такое JBIG2? JBIG2 – это формат, который был создан для сканирования изображений в PDF, когда людям требовалось максимально сильно сжимать PDF файлы из за ограничений интернета и 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. Недостатком является тот факт, что кодировщик может случайно перепутать похожие символы и тем самым изменить суть документа (чеки например).

2. Для улучшения качества сжатых документов кодировщик может хранить референсный символ, а так же, для каждого 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 и SymbolDict). Битовая карта предстовляет собой простой набор символов, а Словарь – это по сути набор карт.

К сегментам можно обращаться по номерам, а специалисный list (Glist) используется для хранениях указателей на все сегменты.

Вернемся к уязвимости. Ее суть заключается в простом переполнении 32-х битного целого числа.

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)

        }

      }

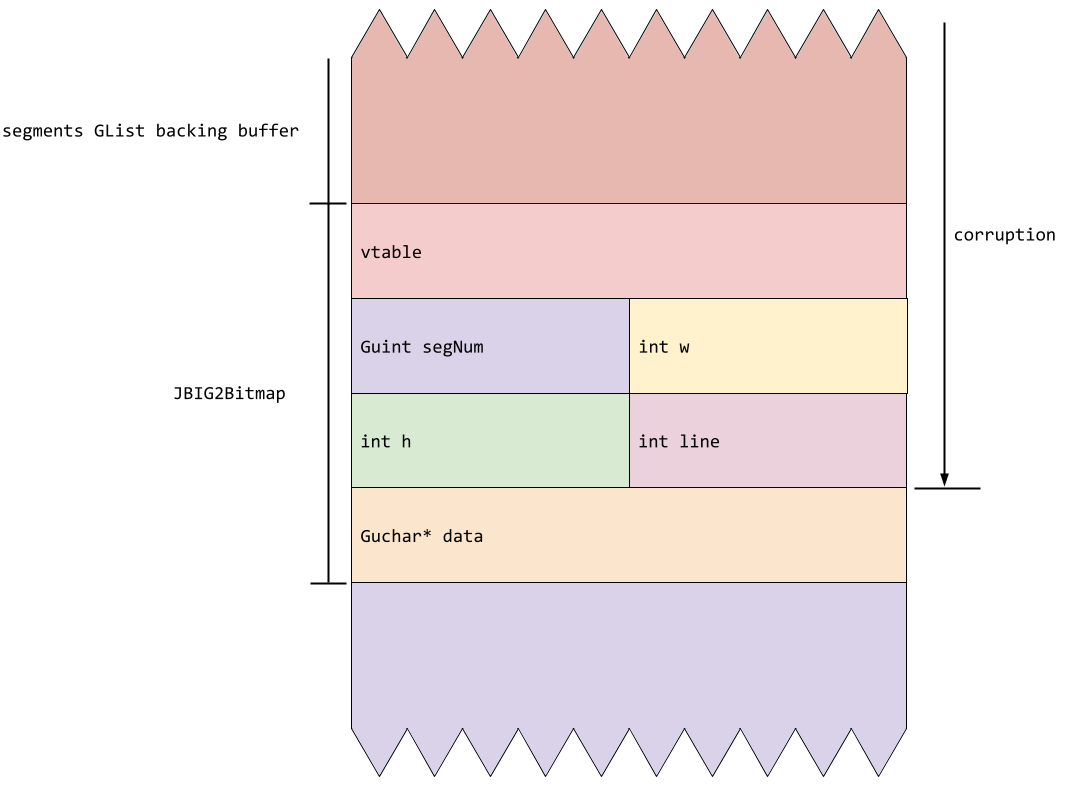
    }

  }

Суть этого кода не очень важна. Нам нужно обратить внимание на переменную numSys (32-х битное целое число). Благодаря заранее подготовленному PDF файлу с кодировкой JBIG2 взлоумышленники добились переполнения в (2) и получили небольшое значение numSys. Это, в свою очередь, вызвало создание буфера слишком маленького размера.

Такое развитие событиый может привезти к записи огромного количества пустых данных и краша обработчика PDF. Но куча обрабатывается так, что первые несколько записей повреждают резервный буфер Glist и перезаписывают указатели на JBIG2Segment на JBIG2Bitmap. Однако такой лавинообразный подход повредил бы всю память, не так ли? Нет, так как Bitmap наследует от Segment виртуальный вызов получения типа сегмента, однако Bitmap возращает другой тип в этом вызове, что и ограничивает дальнейшую запись и защищает злоумышленника от ненужного поверждения памяти.

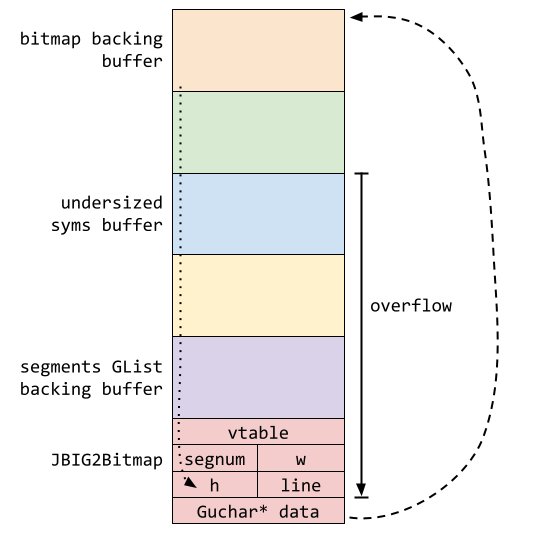
Теперь, когда Glist поврежден, хуцкеры обрабатывают Bitmap, который отвечает за текущую страницу. JBIG2Bitmaps — это простые оболочки вокруг резервного буфера, хранящие ширину и высоту буфера (в битах), а также значение строки, которое определяет, сколько байтов сохраняется для каждой строки.



Далее, злоумышленник переписывает еще 3 указателя Bitmap и останавливает переполнение. При этом указатель vtable и информация, отвечающая за отрисовку текущей страницы, будет переписана. Из-за особенностей структуры адресного пространства iOS эти указатели, скорее всего, будут находиться во вторых 4 ГБ виртуальной памяти с адресами между 0x100000000 и 0x1ffffffff. Поскольку все оборудование iOS имеет прямой порядок байтов (это означает, что поля w и line, скорее всего, будут перезаписаны с помощью 0x1 — наиболее значимой половины указателя JBIG2Bitmap), а поля segNum и h, скорее всего, будут перезаписаны наименее значимой половиной такого указателя, довольно случайное значение, зависящее от структуры кучи и ASLR где-то между 0x100000 и 0xffffffff.

Это дает текущей целевой странице JBIG2Bitmap неизвестное, но очень большое значение h. Поскольку это значение h используется для проверки границ и должно отражать выделенный размер резервного буфера страницы, это приводит к «освобождению» холста рисования. Это означает, что последующие команды сегмента JBIG2 могут читать и записывать память за пределами исходных границ резервного буфера страницы.

Уборщик кучи также размещает резервный буфер текущей страницы чуть ниже буфера syms меньшего размера, так что, когда страница JBIG2Bitmap неограничена, она может читать и записывать свои собственные поля:



Отрисовывая 4-байтовые растровые изображения в правильных координатах холста, они могут записывать во все поля страницы JBIG2Bitmap, а тщательно выбирая новые значения для w, h и line, они могут записывать произвольные смещения из резервного буфера страницы.

Теперь злоумышленник уже сможет переписывать данные по любым адресам, если он будет знать эти адреса. Их можно вычеслить используя смещения из резервного буфера стракницы. 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.

Помните, что JBIG2 умеет работать с различными логическими операторами? А что если использовать их не для работы с глифами, а с отдельными битами? Теперь злоумышленник может создать целые наборы команд. А тк буфер у нас не ограничен, эти команды могут работать с произвольной памятьяю. Обьединяя логические операции в наборы команд, злоумышленники получают возможность создать целую архитектуру и запрограммировать ее, например, для побега из песочницы.

Итак, благодаря целому набору уязвимостей: неограниченные возможности у парсера, банальное переполнение целого числа, а так же отсутсвуию ограничений на какие либо действия со стороны обработчика злоумышленик создал ЦЕЛЫЙ КОМПЬЮТЕР ВНУТРИ КОМПЬЮтера. Такие простые, но в то же время изящные действия смогли показать, насколько нужно быть внимательным при разработке ПО. Ведь даже мелкая ошибка может нанести значительный ущерб безопастности. (заключение говно)

Методы эксплуатации:

Технические Детали Обхода NX (No Execute)

NX защищает систему, запрещая выполнение кода из областей памяти, предназначенных только для данных. Однако, FORCEDENTRY использует технику, известную как Return-Oriented Programming (ROP), для обхода NX. Важным аспектом ROP является использование существующего исполняемого кода в системе для выполнения вредоносных действий, что делает NX неэффективным против такого рода атак: FORCEDENTRY выбирает наборы инструкций (известные как гаджеты), которые уже есть в системе; создает цепочку гаджетов и выполняет нужный код для работы уязвимости.

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: благодаря получению контроля над памятью из за ошибок в разработке алгоиртмов и библиотек, механизм ASLR оказывается неэффективным против этого эксплойта.

Предложения

Улучшение Изоляции и Сегментации

1. Использование аппаратной изоляции: Расширить использование аппаратных средств безопасности для обеспечения дополнительных уровней изоляции критически важных процессов и данных. Это может помочь защитить чувствительные данные и операции даже в случае компрометации операционной системы.

2. Расширенная рандомизация: Улучшение и расширение принципов ASLR для включения более широкой рандомизации и увеличения энтропии расположения ключевых областей памяти, включая рандомизацию стека, кучи, и расположения библиотек в памяти.

3. Ограниение возможностей отдельных библиотек или модулей.

2. Динамический анализ поведения: Внедрение систем динамического анализа поведения программ для выявления и блокировки потенциально вредоносных действий в реальном времени, основываясь на анализе поведенческих паттернов, а не только на сигнатурах вредоносного ПО.

Песочница

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 на хост-систему или другие приложения.

Технические Основы

• Изоляция Процессов: Песочница использует механизмы операционной системы для изоляции процессов, такие как контейнеры или виртуальные машины. Это обеспечивает физическую и логическую изоляцию запущенного кода от остальной системы.

• Контрольный Список Ресурсов: 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.

• Ограничение Прав: Песочница ограничивает права выполнения кода, применяя принцип наименьших привилегий. Это означает, что код выполняется с минимальным набором прав, необходимых для его работы.

Примеры Применения

• Безопасность Браузеров: Современные веб-браузеры используют песочницы для изоляции вкладок и плагинов, предотвращая выполнение вредоносного кода через уязвимости в веб-контенте.

• Мобильные Операционные Системы: iOS и Android широко используют песочницы для изоляции приложений друг от друга и от системных ресурсов, уменьшая тем самым риск вредоносного воздействия на устройство.

Технические Вызовы и Ограничения

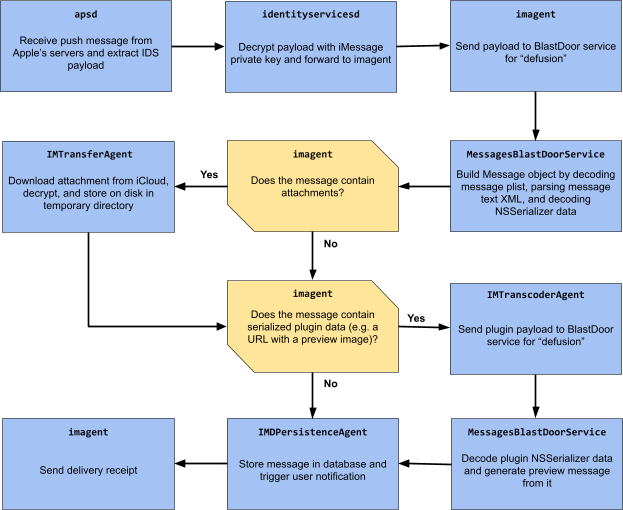
• Производительность: Изоляция и контроль доступа могут влиять на производительность приложений, особенно в средах с интенсивным использованием ресурсов.

• Управление Песочницей: Настройка и управление политиками песочницы требуют глубокого понимания приложений и их потребностей в ресурсах, а также потенциальных угроз безопасности.

• Обход Механизмов Изоляции: Существуют сложные техники и эксплойты, направленные на обход механизмов изоляции песочницы, что требует постоянного анализа угроз и обновления механизмов защиты.

Обзор BlastDoor

BlastDoor является ключевым компонентом системы безопасности, введенным в iOS 14, специально разработанным для обработки входящего контента iMessage. Написан на мемори сейв языке Swift. Этот механизм представляет собой изолированную среду, предназначенную для безопасной обработки и анализа данных 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.



Роль в Защите iMessage

BlastDoor играет центральную роль в стратегии защиты iMessage, обеспечивая несколько уровней защиты. BlastDoor тщательно десериализует входящие сообщения, преобразуя их из формата, используемого для сетевой передачи, в формат, пригодный для обработки и отображения. Этот процесс минимизирует риски, связанные с обработкой ненадежных или специально сформированных данных.

Сендбоксинг в новой версии iOS допускает использование лишь незначительного числа модулей:

only a handful of local IPC services, namely diagnosticd, logd, opendirectoryd, syslogd, and notifyd, can be reached

almost all file system interaction is blocked

any interaction with IOKit drivers (historically a big source of vulnerabilities) is forbidden

outbound network access is denied

До обновления iOS 14 у Apple была серьезная архитектурная проблема, связанная с ASLR. Большинство библиотек, используемых системой, были доступны заранее, причем адрес рандомизировался только один раз, при запуске системы. Теперь же, когда какой то сервис пытается получить доступ к кешу библиотек, адрес этого участка памяти рандомизируется.

Еще одним важным введением стало намеренное замедление доступа к определенному сервису после его краша. Это обновление значительно снижает скорость работы эксплойтов, которые пытались получить доступ к сервису с помощью банального перебора.

Концепция "Побега из Песочницы"

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 злому хакеры использовали (насрать про NSExpression and NSPredicate и Prior to iOS 14.5 the isa field of an Objective-C object was not protected by Pointer Authentication Codes (PAC), so the JBIG2 machine builds a fake Objective-C object with a fake isa such that the invocation of the dealloc selector causes the deserialization and evaluation of the NSFunctionExpression. This is very similar to the technique used by [Samuel in the 2020 SLOP post](https://googleprojectzero.blogspot.com/2020/01/remote-iphone-exploitation-part-3.html).)

На прошлом шаге злоумышленники создали набор NSFunctionExpression. Этот обьект нужен для двух вещей: 1) замести следы, удаляя те самые гифки, из за которых все началось; 2) создание пейлода и получение Pegasus’а.

Поговорим про заметание следов. Шпионское ПО на то и шпионское, чтобы остаться незамеченным. Функциональное выражение получает доступ ко всем временным файлам, выбирает гифки, которые были исполоьзованны для взлома и удаляет их. Просто и элегантно.

Теперь идем дальше. Для вызова методов из других процессов в Objective C используятся такой механизм, как NSXPConnection (semi transparent rpc mechanism). Важно отметить, что для “пересылки” обьектов через этот механизм необходимо, чтобы они имели свои протоколы (описывающие доступные для вызова методы). Для сериализации доступны подклассы любых классов без ограничения на наследственнсоть. Это означает, что любой обьект протокола, который обьявляет определенный тип поля, так же принимает любой подкласс этого типа. Таким образом, протокол допускает использование любого подкласса (а это уже является критической угрозой безопасности).

Теперь можно передать Array состоящий из двух элементов: обьекта синтеза голоса (странно, да?) и обьекта PrototypeToolSection с одним рядом PTRow. Но ведь для использования прототипа нам наверняка нужны соответсвующие библиотеки. Для этого и нужен синтез голоса. Когда мы передаем обьект 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.

Что же тогда нас ждет в PTRow? Старый добрый предикат. Его 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!).

Такой же подход используяется для “выполнения” еще несколько пейлодов в контексте модуля, отвечающего за связь. Они делают следующее:

1. Создания url по определенному адресу и сохрание его
2. Подгрузка дополнительных библиотек
3. Параметризация url из первого шага с указанием точной модели телефона
4. Создание полного URL адреса и получение данных сервера. Расшифровка данных с помощью специального ключа.

Таким образом, злоумышленник смог сбежать из сендбокса и запросить следующую информцаию с управляющего сервера и выполнить указания. В этот момент уже становится ясно, что хакеры нанесли серьезный ущерб с точки зрения безопасности, воспользовавшись большим количеством ошибок в архитектуре.

Заключение

Итак, главной особенностью побега является не какие то специальные методики по взлому, а банальное использование дыр в ПО.

И все это работает не на какой то специфичной версии ПО; оно даже никак не повреждает память, как будто так и должно было все работать.

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

Недостатки в Обработке Входных Данных

FORCEDENTRY использовал специально сформированные файлы, чтобы эксплуатировать уязвимости в обработке изображений и других медиа-форматов внутри iMessage. Эти файлы были разработаны так, чтобы вызвать непредвиденное поведение в программном обеспечении, что часто является результатом:

Переполнения Буфера: Когда данные превышают выделенный для них буфер, что может привести к перезаписи важных участков памяти.

Ошибки Десериализации: Неправильная обработка входных данных может привести к выполнению произвольного кода при десериализации объектов.

Использование Логических Ошибок

Атака использовала логические ошибки в программном обеспечении, такие как недостаточная проверка состояний или условий, под которыми могут быть обработаны данные, приводя к несанкционированному выполнению кода.

Обход Механизмов Защиты

Обход ASLR и DEP: FORCEDENTRY демонстрирует, как атакующие могут использовать сложные методы для обхода механизмов защиты, таких как Address Space Layout Randomization (ASLR)

Недостающая изоляция. Возможность определенных модулей взаимодействовать с теми, которые джае в теории им не будут нужны, а так же доступ к различным специфичным библиотекам могут привести к нарушению безопасности.

Данный эксплоит продемонстрировал, что ошибки, которые выглядят мелко и незначительно по отдельности, способны нанести серьезный ущерб и скомпрометировать такое сложное устройство, как смартфон. Хакеры собрали из них сложную систему, которая смогла не только собрать полноценный компьютер в буфере, но и выйти из сендбокса и скачать шпионское ПО.

Помимо этих, распространенными являются следующие ошибки:

1. Недостаточная валидация входных данных (например sql иньекции, xss атаки)
2. Ошибки управления памятью. Например, Use-After-Free: обращение к памяти после того как она освобождена может привести к выполнению вредоносного кода, если хакер имеет доступ к ней.
3. Недостаточная аутентификация: слабые политики аутентификации могут позволить получить доступ к учетным данным пользователей. Из за проблем с авторизацией пользователи могут получить доступ к фунциям, которые должны быть недоступны для них
4. Неправильная обработка ошибок: если приложение возвращает подробные сообщения об ошибках пользователю, это может предоставить атакующим ценную информацию о внутреннем устройстве системы.
5. Отказ в обслуживании: Неправильная обработка исключений может привести к сбоям в работе приложения, что может быть использовано для атак типа отказ в обслуживании (DoS).

Заключение

Случай FORCEDENTRY подчеркивает важность комплексного подхода к безопасности программного обеспечения, включая тщательное проектирование, регулярное тестирование и быстрое реагирование на обнаруженные уязвимости. Это демонстрирует, что даже в высокозащищенных системах могут существовать сложные уязвимости, которые могут нанести серьезный ущерб безопасности.