God Mode On

Исследователи запустили Doom на головном устройстве автомобиля, удаленно атаковав его модем

Александр Козлов
Сергей Ануфриенко

13.11.2025 Версия 1.0

Введение ... 3
Получение встроенного программного обеспечения модема ... 5
Удаленный доступ в модем (CVE-2024-39431) ... 6
Закрепление в системе ... 9
Lateral Movement внутри SoC ... 13
Разработка эксплойта под AP ... 16
Этап 1: Поиск базового адреса ядра Linux ... 17
Этап 2: Локализация таблицы kallsyms ... 17
Этап 3: Выбор системного вызова для перехвата ... 18
Этап 4: Локализация функции call_usermodehelper ... 19
Этап 5: Отключение SELinux ... 19
Этап 6: Поиск области для внедрения кода ... 19
Этап 7: Создание и внедрение шелл-кода ... 20
Этап 8: Модификация таблицы системных вызовов ... 21
Заключение ... 23

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Эксплуатация уязвимости, обнаруженной в модеме, который устанавливается в головное устройство некоторых автомобилей, позволила экспертам Kaspersky ICS CERT получить полный контроль над системой.

Введение

Представьте себя водителем, который мчится по автостраде за рулем своего новенького электромобиля. И вдруг видит, что вместо карты с маршрутом или меню управления различными функциями на весь огромный мультимедийный дисплей запускается Doom – культовый 3D-шутер всех времен и народов, – и осознает, что кто-то в эту игру прямо сейчас пытается играть, управляя персонажем удаленно. Это не сон и не разгулявшаяся фантазия, а вполне достижимая в современных реалиях ситуация, что более чем наглядно продемонстрировали эксперты Kaspersky ICS CERT.

Мы знаем, что интернет вещей играет важную роль в современном мире, где к Сети подключены не только смартфоны и ноутбуки, но и заводы, автомобили, поезда, и даже самолеты. Большую часть времени подключение осуществляется через мобильные сети передачи данных 3G/4G/5G с помощью модемов, установленных в этих транспортных средствах и устройствах. При этом все чаще модемы интегрируются в систему на кристалле (System on Chip, SoC), которая способна выполнять одновременно несколько функций, используя модемный процессор (Communication Processor, CP) и процессор приложений (Application Processor, AP). Операционная система общего назначения, такая как Android, вполне может работать на AP, в то время как CP, предназначенный для взаимодействия с мобильной сетью, обычно реализуется на базе специализированных ОС. При этом взаимосвязь между AP, CP и RAM на этом кристалле на уровне микроархитектуры представляет собой «черный ящик» и известна только производителю, хотя от этого зависит безопасность всей SoC.

Считается, что обход механизмов безопасности 3G/LTE – это исключительно академическая задача, поскольку при подключении пользовательского устройства (User Equipment, UE) к базовой станции сотовой связи (Evolved Node B, eNB) устанавливается безопасный канал связи. Даже если кто-то сможет обойти эти механизмы, обнаружить уязвимость в модеме и выполнить на нем свой код, это скорее всего не поставит под угрозу бизнес-логику устройства. Эта логика (например, пользовательские приложения, история браузера, звонки и SMS на

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

смартфоне) находится на AP и предположительно не может быть доступна с модема. Или может?

Чтобы выяснить это, мы провели исследование безопасности современной SoC Unisoc UIS7862A, оснащенной встроенным 2G/3G/4G-модемом. Такую SoC можно встретить в китайских и отечественных мобильных устройствах или, что более интересно, в головных устройствах современных китайских автомобилей, которых на дорогах страны становится все больше. Головное устройство – один из ключевых объектов автомобиля, и нарушение его информационной безопасности ставит под угрозу не только сохранность пользовательских данных, но и безопасность дорожного движения.

В ходе своего исследования мы обнаружили несколько критических уязвимостей на разных уровнях стека сотовых протоколов модема Unisoc UIS7862A. В этой статье речь идет об уязвимости переполнения стека в реализации протокола 3G RLC (CVE-2024-39432). Эта уязвимость может быть использована для удаленного выполнения кода на ранних этапах подключения, причем до активации каких-либо защитных механизмов.

При этом получение возможности выполнения своего кода на модеме – лишь точка входа для полной удаленной компрометации всего SoC. Наши дальнейшие усилия были направлены на получение доступа к AP. Мы обнаружили несколько способов получить такой доступ, в том числе с использованием аппаратной уязвимости в виде скрытого периферийного DMA-устройства (Direct Memory Access) для выполнения lateral movement внутри SoC. Это позволило нам установить собственный патч к работающему ядру Android и выполнить произвольный код на AP с наивысшими привилегиями. Подробности представлены в соответствующих разделах.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Получение встроенного программного обеспечения модема

Модем, ставший предметом нашего исследования, мы обнаружили на плате головного устройства одного китайского автомобиля.

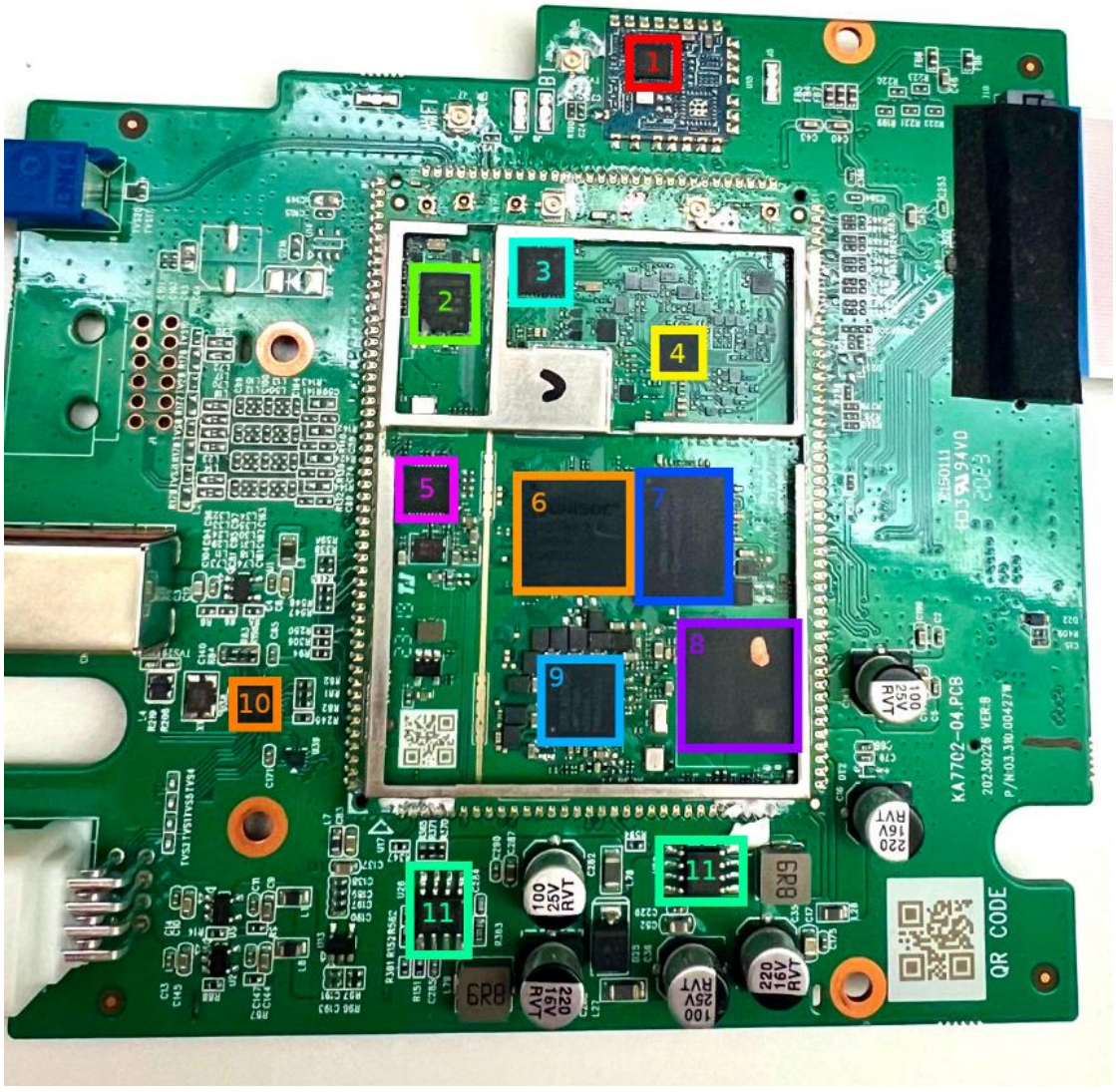


Рисунок – Плата головного устройства

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Таблица – Описание компонентов платы

Номер на фото	Наименование
1	Контроллер Realtek RTL8761ATV 802.11b/g/n 2.4G с интерфейсами Wireless LAN (WLAN) и USB (стандарты USB 1.0/1.1/2.0)
2	Микросхема SPRD UMW2652 BGA WiFi
3	Микросхема 55966 TYADZ 21086
4	Радиочастотный трансивер SPRD SR3595D (Unisoc)
5	Видеодекодер Techpoint TP9950
6	UNISOC UIS7862A
7	Внутренний накопитель BIWIN BWSRGX32H2A-48G-X, Package200-FBGA, ROM Type – Discrete, ROM Size – LPDDR4X, 48G
8	Карта памяти SCY E128CYNT2ABE00 EMMC 128G/JEDEC
9	Контроллер питания SPREADTRUM UMP510G5
10	Чип-шунт FEI.1s LE330315 USB2.0
11	Синхронный понижающий DC-DC преобразователь с внутренней компенсацией SCT2432STER

Используя сведения об аппаратном устройстве модема, мы выпаяли и считали встроенную мультимедийную карту памяти, содержащую полный образ его операционной системы. Затем мы проанализировали полученный образ.

Удаленный доступ в модем (CVE-2024-39431)

Исследуемый модем, как и любой современный модем, реализует нескольких стеков протоколов: 2G, 3G, LTE. Очевидно, чем больше протоколов поддерживает устройство, тем больше потенциальных точек входа (векторов атаки) оно имеет. При этом чем ниже в стеке сетевой модели OSI находится уязвимость, тем серьезнее последствия ее эксплуатации, поэтому мы решили провести анализ механизмов фрагментации пакетов данных на уровне доступа к среде передачи (протокол RLC).

Нас заинтересовал именно этот протокол, потому что он используется для установки безопасного зашифрованного канала передачи данных между базовой станцией и модемом, по нему, в частности, передаются данные протокола NAS (Non-Access Stratum). Таким образом, наличие уязвимости

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

удаленного выполнения кода (Remote Code Execution, RCE) позволяет исполнить свой код на модеме в обход всех существующих механизмов защиты коммуникации в 3G.

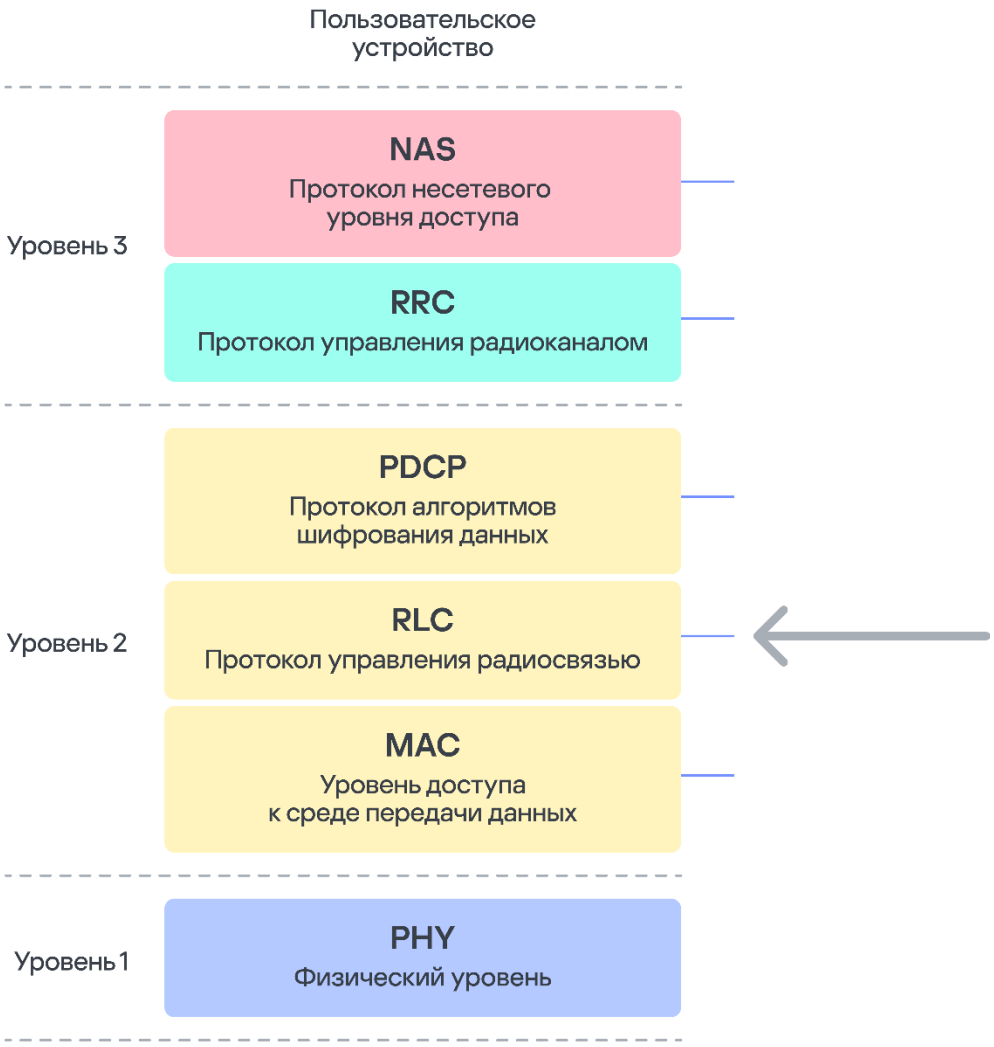


Рисунок – Стек протоколов 3G

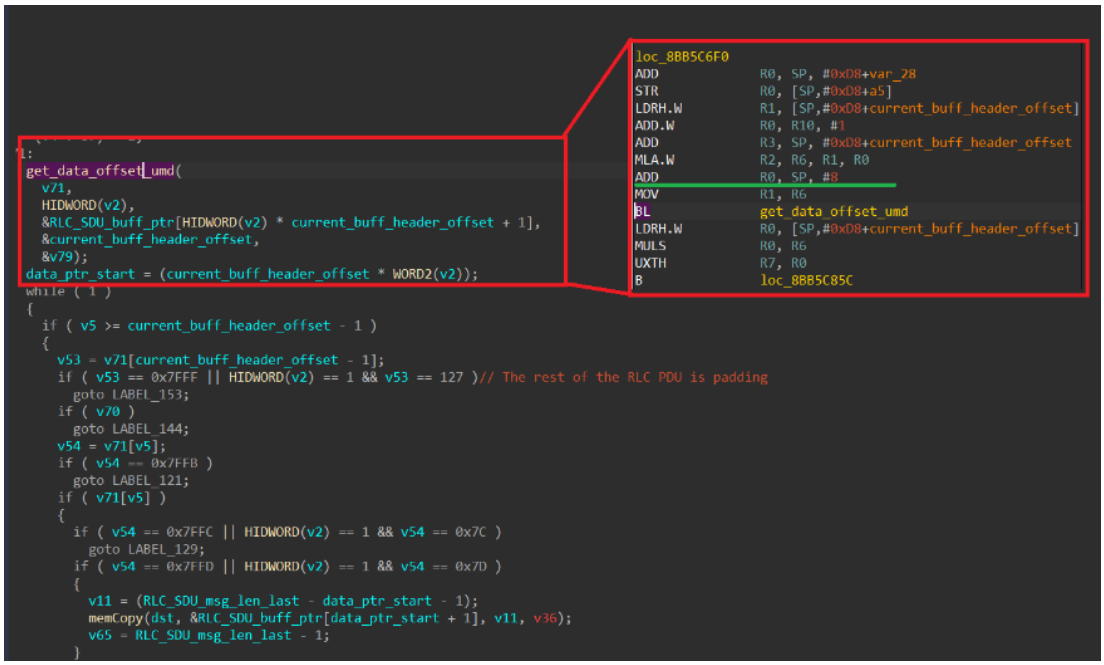
Протокол RLC использует три различных режима передачи: TM, UM и AM. Нас интересует только режим UM (Unacknowledged Mode), потому что в нем стандарт 3G предусматривает как сегментацию данных, так и конкатенацию нескольких небольших фрагментов высокоуровневых данных (Protocol Data Unit, PDU) в один фрейм канального уровня. Это сделано для максимальной

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

утилизации канала передачи. На уровне RLC пакеты называются блоками служебных данных (Service Data Unit, SDU)[1].

Чтобы найти в прошивке функции обработки SDU среди огромного количества различных функций (примерно 75 000), достаточно поискать константы, используемые в процессе работы (0x7FFF, 0x7FFC, 0x7FFB). Среди всех мест, где встречаются эти константы, мы выделяем те, в которых они осмысленно применяются в контексте исполнения кода. В итоге получаем ограниченный набор функций, включая функцию обработки входящего SDU-пакета.

При обработке принятого SDU-пакета происходит разбор полей его заголовка. Сам пакет состоит из обязательного заголовка, необязательных заголовков и данных. Обработка необязательного заголовка заключается в последовательном проходе по каждому полю. Для этого используется функция get_data_offset. Она разбирает необязательные заголовки, собирает размеры фрагментов данных внутри части данных и возвращает указатель на данные внутри SDU-пакета. Важно: информация о размере всех конкатенированных PDU внутри SDU, согласно анализу кода операционной системы модема, хранится на стеке!



Алгоритм последовательно обрабатывает каждое поле заголовка. Признаком конца необязательных заголовков служит младший бит (E bit), равный 0. Если же он равен 1, то обработка продолжается. В процессе обработки данные записываются в переменную, находящуюся на стеке

[1] См. стандарт ETSI TS 136 322.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

вызывающей функции. При этом количество необязательных заголовков не ограничено. Глубина стека составляет 0xB4 байт. Размер пакета, который можно парсить (количество заголовков, каждый заголовок – это запись 2 байт на стек), ограничен размером SDU-пакета в 0x5F0 байт.

В итоге эксплуатация возможна всего лишь одним пакетом, в котором количество заголовков превышает глубину стека (90 заголовков). Важно отметить, что именно в этой функции стековая канарейка отсутствует, и при переполнении стека как раз в этой функции можно подменить адрес возврата и значения некоторых non-volatile регистров. Однако подмена возможна только на значение, оканчивающееся на единицу в бинарном виде (младший бит равен 1). Примечательно, что исполнение происходит на ARM в режиме Thumb, поэтому все адреса возврата должны иметь младший бит, равный 1. Совпадение? Возможно.

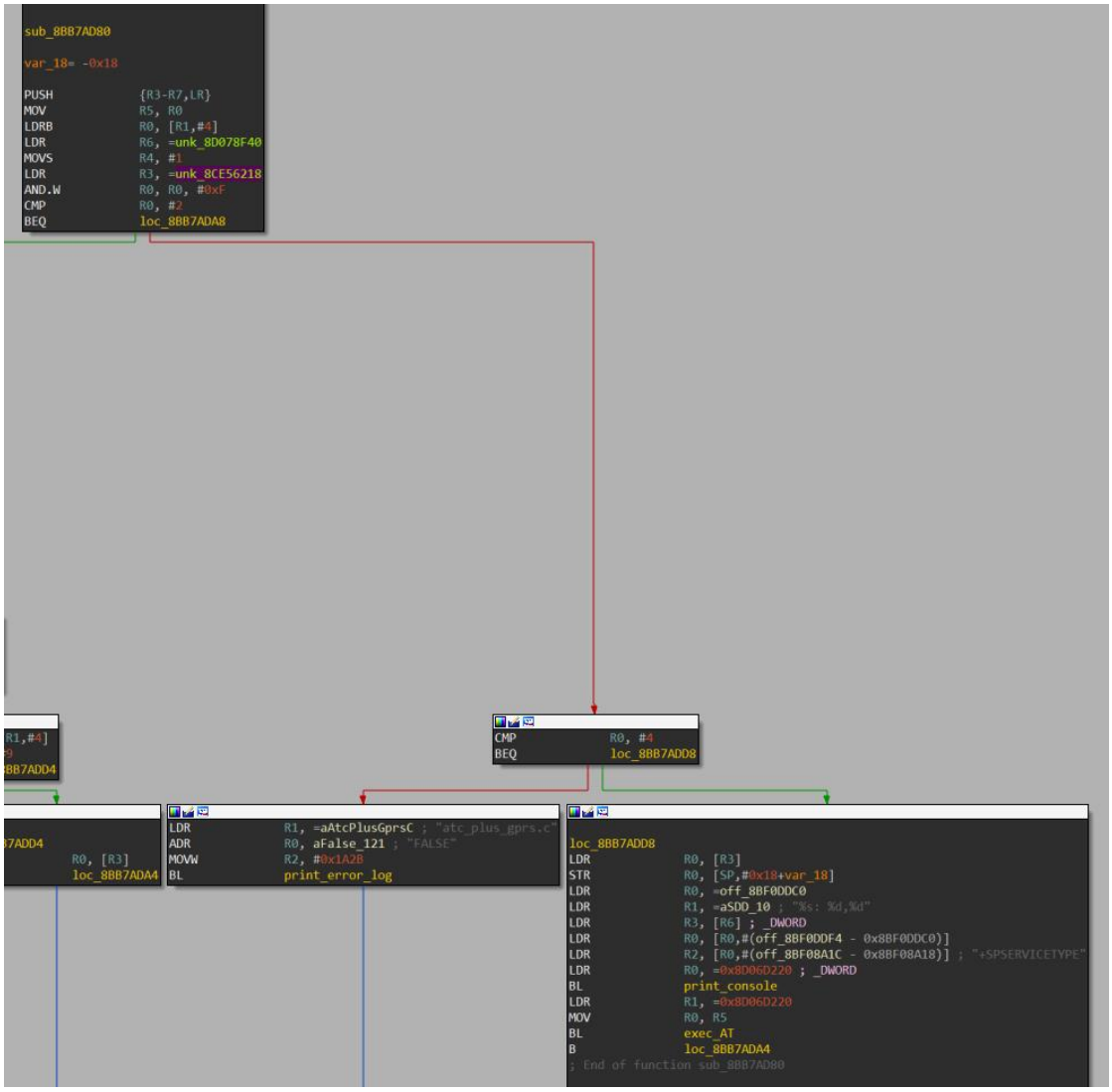
Так или иначе, первая же отправка dumm-пакета SDU с соответствующим количеством «правильных» заголовков привела к перезагрузке устройства. Однако на тот момент у нас не было возможности получить информацию о том, где и почему произошло падение (хотя мы предполагаем, что причиной стала попытка передачи управления по адресу 0xAABBCCDD, взятому из нашего пакета). Посмотрим, что с этим можно сделать.

Закрепление в системе

Первое и самое важное наблюдение: мы знаем, что указатель на только что полученный SDU-пакет хранится в регистре R2. Можно использовать технику возвратно-ориентированного программирования (Return Oriented Programming, ROP), чтобы выполнить свой код. Однако остается вопрос: можем ли мы вообще исполнить свой код?

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Чтобы убедиться в том, что мы действительно можем исполнить свой код на стороне модема, воспользуемся доступным обработчиком AT-команд. Поскольку нам неизвестен текущий адрес фрейма стека, на котором лежат наши данные, неизвестно, исполняем ли стек вообще, и мы не уверены, доступны ли секции кода для записи (хотя догадываемся, что нет), то самый надежный способ – переписать данные в области RAM-памяти. Для этого ищем подходящую функцию среди доступных AT-команд. Первая подходящая команда – SPSERVICETYPE.



Далее необходимо использовать ROP-гаджеты, чтобы переписать адрес 0x8CE56218, не нарушая при этом дальнейшую работу алгоритма обработки входящих SDU-пакетов. Для этого достаточно обеспечить возврат в функцию, из которой был вызван обработчик SDU-пакета, потому что она вызывается как callback, а значит, отсутствует связь по данным на стеке.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Учитывая, что эта функция добавила на стек всего 0x2C байт, необходимо уложиться в этот размер.

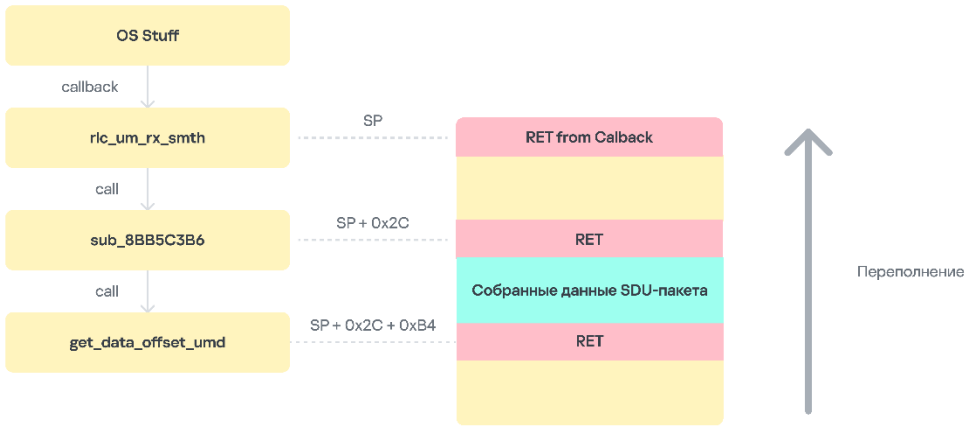


Рисунок – Переполнение стека в контексте операционной системы

Найдя подходящую ROP-цепочку, мы запускаем SDU-пакет, содержащий ее в качестве полезной нагрузки. В результате в консоли AT-команды SPSERVICETYPE видим вывод 0xAABBCCDD. Наш код работает!

```
at /dev/sttu_lte5 & sleep 0.1; echo -e "AT+SPSERVICETYPE?\r" > /dev/stty_lte5 *


+SPSERVICETYPE: 0,2864434397


OK
console:/ #
```

Далее, по аналогии, мы вводим адрес фрейма стека, на котором находятся наши данные. Однако, к сожалению, оказывается, что стек не является исполняемым. Теперь перед нами стоит задача разобраться с настройкой MPU на модеме. Для этого, снова используя метод ROP-цепочек, генерируем код, который будет читать таблицу MPU по DWORD за раз. Спустя множество итераций мы получаем следующую таблицу.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

```
Base Address: 00 00 00 00   Size and Regs: 00 00 00 3F   ACR: 00 00 10 00 (0x00000000) 4GB
Base Address: 00 00 00 00   Size and Regs: 00 00 00 3D   ACR: 00 00 11 10 (0x00000000) 2GB

Base Address: 00 00 00 00   Size and Regs: 00 00 00 17   ACR: 00 00 00 10 (0x00000000) 4Kb
Base Address: 00 00 00 00   Size and Regs: 00 00 00 13   ACR: 00 00 00 50 (0x00000000) 1Kb
Base Address: 00 00 00 C0   Size and Regs: 00 00 00 13   ACR: 00 00 00 10 (0x000000C0) 1Kb

Base Address: 87 80 00 00   Size and Regs: 00 00 00 2D   ACR: 00 00 00 10 (0x87800000) 8Mb
Base Address: 88 00 00 00   Size and Regs: 00 00 00 33   ACR: 00 00 00 10 (0x88000000) 64Mb
Base Address: 88 00 00 00   Size and Regs: 00 00 E0 31   ACR: 00 00 10 00 (0x88000000) 32Mb (sub-region)
Base Address: 89 40 00 00   Size and Regs: 00 00 00 29   ACR: 00 00 10 00 (0x89400000) 2Mb
Base Address: 8B 00 00 00   Size and Regs: 00 00 00 2F   ACR: 00 00 00 50 (0x8b000000) 16Mb (CODE SECTION)
Base Address: 8C 00 00 00   Size and Regs: 00 00 80 31   ACR: 00 00 00 10 (0x8C000000) 32Mb (sub-region)
Base Address: 8D 80 00 00   Size and Regs: 00 00 F1 2D   ACR: 00 00 00 10 (0x8D800000) 8Mb (sub-region)

Base Address: 00 00 00 00   Size and Regs: 00 00 00 00   ACR: 00 00 00 00 (unused)
Base Address: 00 00 00 00   Size and Regs: 00 00 00 00   ACR: 00 00 00 00 (unused)
Base Address: 00 00 00 00   Size and Regs: 00 00 00 00   ACR: 00 00 00 00 (unused)
Base Address: 00 00 00 00   Size and Regs: 00 00 00 00   ACR: 00 00 00 00 (unused)
```

По таблице видно, что секция кода замаплена, как мы и предполагали, только на исполнение. Попытка изменить настройку приводит к появлению еще одной ROP-цепочки, только теперь в таблице, на неиспользуемом месте, эта же секция замаплена с правами записи. Это возможно благодаря особенностям программирования MPU, а именно наличию механизма overlap и тому факту, что регион с большим ID имеет более высокий приоритет.

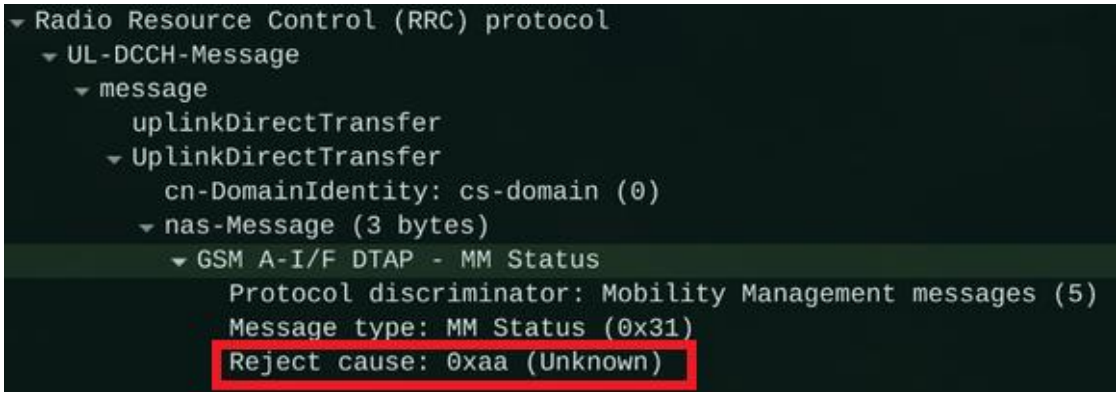
```
Base Address: 8B 00 00 00   Size and Regs: 00 00 00 2F   ACR: 00 00 03 08 (0x8B000000) 16Mb
```

Остается воспользоваться указателем на наши данные (он все еще лежит в R2) и пропатчить только что разблокированную на запись секцию кода. Вопрос в том, что именно патчить. Самый простой путь – пропатчить обработчик протокола NAS, добавив в него свой код. Для этого воспользуемся одной из команд протокола NAS – MM information[2]. С ее помощью можно за один раз переслать большой объем данных, а в ответ получить один байт данных по команде MM status.

В итоге нам удается не только получить возможность исполнения своего кода на стороне модема, но и установить с ним полноценную двухстороннюю связь, используя высокоуровневый протокол NAS в качестве способа доставки сообщений. В данном случае это пакет MM Status с полем cause, равным 0xAA.

[2] См. стандарт ETSI TS 124 008, стр. 393.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»



Однако исполнение своего кода на стороне модема не позволяет получить доступ к пользовательским данным. Или позволяет?

Lateral Movement внутри SoC

Исследование внутреннего устройства модемного процессора в контексте микроархитектуры SoC выявляет множество потенциальных векторов атак на AP. При анализе внутреннего устройства модема мы сразу заметили, что в нем используются реальные, а не виртуальные адреса RAM. Кроме того, наше внимание привлекли адреса, по которым располагалось ядро операционной системы AP.

```
ums512_1h10:/data/local/tmp # cat /proc/iomem
.............................................
70c00000-70c00fff : /soc/ap-apb/i2s@70c00000
70d00000-70d00fff : /soc/ap-apb/i2s@70d00000
70e00000-70e00fff : /soc/ap-apb/i2s@70e00000
71100000-71100fff : /soc/ap-apb/sdio@71100000
71200000-71200fff : /soc/ap-apb/sdio@71200000
71400000-71400fff : /soc/ap-apb/sdio@71400000
71700000-7170ffff : /soc/ap-ahb/vdsp
80000000-1fffefff : System RAM
  80080000-80d8ffff : Kernel code
  81010000-819cefff : Kernel data
```

Возникает гипотеза: что, если CP и AP используют одно и то же адресное пространство, ведь оперативная память в SoC, вероятно, реализована как единый аппаратный компонент? Анализ Device Tree на стороне AP еще больше укрепил предположение, что физическое адресное пространство у

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

CP и AP, скорее всего, совпадает (сравните с таблицей MPU, представленной выше).

```
cp-mem at 0x89600000 size 0x4600000
iq-mem at 0x90000000 size 0x4000000
sml-mem at 0x94000000 size 0x20000
debug-mem at 0x100000000 size 0x1000
sipc-mem at 0x87800000 size 0x800000
```

Остается лишь проверить достоверность гипотезы. Для этого нужно снова пропатчить таблицу MPU, на это раз добавив запись, которая разрешает доступ на чтение и запись к памяти, начиная с адреса 0x80000000. В результате нам удается замапить адресное пространство AP в адресное пространство CP. В качестве доказательства концепции мы патчим первую страницу ядра Linux.

```
unisoc_via_python python3 py_usb.py -r 0x80080000
00000000: 00 40 34 14 00 00 00 00  00 00 08 00 00 00 00 00  .@4.............
00000010: 00 F0 19 00 00 00 00 00  0A 00 00 00 00 00 00 00  ................
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  70 77 6E 64 00 00 00 00  ........pwnd....
00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Казалось бы, на этом можно остановиться: теперь у нас есть возможность исполнения своего кода не только на модеме, но и на AP. Но остается вопрос: а есть ли другой способ?

В поисках ответа мы решили исследовать доступную аппаратную периферию на стороне модема. Особый интерес для нас представляет контроллер DMA. Анализ кода показывает, что все контроллеры DMA располагаются в области памяти модема, начиная с адреса 0x20000000. Однако любая попытка чтения памяти по этим адресам приводит к DataAbort на уровне аппаратного ядра. В чем причина?

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Анализируем код, отвечающий за работу с контроллером DMA. Примечательно, что некоторые его участки, не имеют вызовов, будто их забыли удалить из релизной версии операционной системы модема. Используя фрагменты «забытого» кода, мы получаем доступ к контроллеру DMA. Оказывается, большая часть периферии физически отключена от питания. Для ее включения необходимо выполнить запись в специальный аппаратный регистр.

```
LDR     R3, =0x20071034  //unclock DMA
LDR     R2, =0x00000002
STR     R2, [R3]
```

Помимо включения, для работы контроллера DMA нужно также подать на него тактовый сигнал clock. За это отвечает другой аппаратный регистр в том же регионе памяти.

```
LDR     R3, =0x20071060 // enable DMA Clock
LDR     R2, =0x00000001
STR     R2, [R3]
```

В итоге нам удается разблокировать доступ к неиспользуемому в операционной системе модема контроллеру DMA. С его помощью мы, точно так же, как ранее через таблицы MPU, перезаписываем первую страницу операционной системы AP.

```
ums512_1h10:/ #
ums512_1h10:/ # /data/local/tmp/mem-rw-app -x -m -r 0x80080000 128
Going to operate on address: 80080000 and device /proc/rmem
Length: 128 (0x80) bytes
0000:   00 40 34 14  00 00 00 00  00 00 08 00  00 00 00 00   .@4.............
0010:   00 f0 94 01  00 00 00 00  0a 00 00 00  00 00 00 00   ................
0020:   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
0030:   00 00 00 00  00 00 00 00  64 6d 61 20  70 77 6e 64   ........dma pwnd
0040:   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
0050:   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
```

К сожалению, в отличии от таблиц MPU, программное обновление не защищает от этой микроархитектурной особенности. Очевидно, предоставляемые ею возможности позволяют нарушить безопасность AP, имея всего лишь одну уязвимость уровня RCE на модеме.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Разработка эксплойта под AP

После получения возможности извне модифицировать оперативную память AP (либо посредством исполнения кода на модеме и перенастройки его MPU, либо с помощью DMA-периферии) перед нами встает вопрос: как исполнить свой код на AP и закрепиться в его операционной системе, имея лишь возможность читать и писать его память?

Для демонстрации (Proof of Concept) в качестве полезной нагрузки мы пробуем установить и запустить на AP игру Doom. Реализуемая схема выглядит следующим образом: часть полезной нагрузки, работающая на модеме, последовательно локализует ключевые структуры ядра Linux, после чего внедряет в него собственный код для выполнения необходимых действий. Весь процесс должен происходить без участия пользователя и обходить стандартные механизмы безопасности Android.

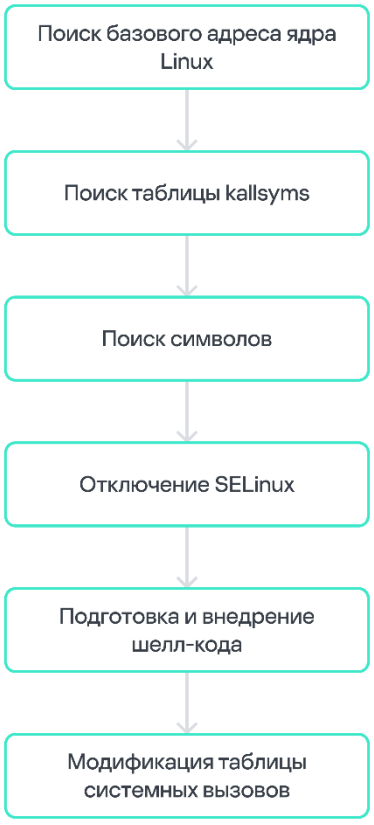


Рисунок – Алгоритм исполнения своего кода на стороне AP

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Этап 1: Поиск базового адреса ядра Linux

Первым шагом атаки является определение базового адреса загруженного в память ядра Linux. В современных версиях Android ядро часто использует технологию рандомизации адресного пространства (Address Space Layout Randomisation, ASLR), тем не менее его базовый адрес легко найти с помощью поиска по известным сигнатурам. В нашем случае это даже не требуется, поскольку базовый адрес ядра всегда равен 0x80080000 (PA) или 0xffffff8008080000 (VA).

Этап 2: Локализация таблицы kallsyms

После обнаружения базового адреса ядра необходимо найти таблицу символов ядра (kallsyms). Эта таблица содержит адреса всех экспортируемых функций и переменных ядра, что позволяет обнаружить адреса требуемых для проведения атаки объектов.

Таблица kallsyms обычно располагается в секции .rodata и имеет характерную структуру:

- массив адресов символов;
- массив имен символов;
- индексная таблица;
- таблица типов символов.

При этом массив адресов символов может использовать относительную адресацию вместо абсолютной для экономии памяти на 64-битных системах, а строки с именами символов могут быть дополнительно сжаты. В нашем случае оба этих механизма (относительная адресация и сжатие имен символов) были использованы, что затрудняло поиск таблицы в памяти ядра. Тем не менее, таблицу удалось найти с использованием эвристического алгоритма, который «собирает» ее по данным, которые считает правильными, и проверяет целостность после сборки.

В нашем случае нужно найти следующие критически важные для проведения атаки символы:

- sys_call_table – таблица системных вызовов;
- call_usermodehelper – функция запуска пользовательских процессов из ядра;
- selinux_enforcing – флаг состояния SELinux.

Для распаковки имен символов ядра воспользуемся следующим кодом.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

```rust
fn extract_name<'b>(&self, pos: usize) -> (&'b str, usize) {
    const MAX_NAME_LEN: usize = 256; // Adjust the buffer size as needed
    let name_len = self.names[pos] as usize;
    let mut buffer_index = 0;
    let mut off = pos + 1;
    let mut skipped_first = false;

    unsafe {
        for _ in 0..name_len {
            let token_index = self.names[off] as usize;
            off += 1;
            let token = self.get_token(token_index);

            for &byte in token.as_bytes() {
                if !skipped_first {
                    skipped_first = true;
                } else if buffer_index < MAX_NAME_LEN {
                    BUFFER[buffer_index] = byte;
                    buffer_index += 1;
                }
            }
        }

        let result_str = str::from_utf8(&BUFFER[..buffer_index]).unwrap_or("");
        (result_str, name_len + 1)
    }
}

fn get_token(&self, token_index: usize) -> &str {
    let start = self.token_index[token_index] as usize;
    let end = self.token_table[start..]
        .iter()
        .position(|&c| c == 0)
        .map(|p| start + p)
        .unwrap_or_else(|| self.token_table.len());
    str::from_utf8(&self.token_table[start..end]).unwrap()
}
```

Этап 3: Выбор системного вызова для перехвата

Таблица системных вызовов (sys_call_table) является ключевым элементом для атаки. В нашей системе она расположена по адресу 0x809D2000 (PA) или 0xffffff80089d2000 (VA). Эта таблица представляет собой массив указателей на функции-обработчики системных вызовов. Каждый элемент таблицы соответствует определенному системному вызову, идентифицируемому номером. Эти номера – фиксированные для каждой

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

архитектуры, для ARM64 их можно посмотреть, например, на Chromium OS Docs.

Нас интересует системный вызов getpriority с номером 141 (0x8d). Соответствующий элемент в таблице находится по смещению 141*8=0x468=1128 байт. Таким образом, указатель на обработчик getpriority расположен по адресу 0x809D2468 (PA).

Этап 4: Локализация функции call_usermodehelper

Получение адреса фукнкции call_usermodehelper после обнаружения и реконструкции таблицы символов ядра (kallsyms) становится простой задачей. В нашей системе эта функция располагается по адресу 0xffffff80080bfe00. Она позволяет драйверам ядра запускать пользовательские процессы, и именно ее мы будем использовать для загрузки, установки и запуска APK-файла с Doom.

Этап 5: Отключение SELinux

Для успешного выполнения атаки необходимо временно (или насовсем) отключить механизм SELinux, иначе он будет препятствовать запуску пользовательских процессов с помощью User Mode Helper. Для этого мы ищем в таблице символов ядра адрес глобальной переменной selinux_enforcing. Установив для нее значение 0, отключаем принудительное применение политик SELinux.

Этап 6: Поиск области для внедрения кода

Теперь нужно найти подходящее место в памяти ядра для размещения шелл-кода. Это должна быть неиспользуемая «дырка» в секции кода ядра, где можно разместить дополнительный код, не нарушая работу системы. В нашей системе секция кода занимает область от 0x80080800 до 0x809d0000 и имеет размер 0x94f800 байт. Путем статического анализа обнаруживаем подходящую по размеру область свободной памяти по адресу 0x809cb000 – ближе к концу секции кода ядра. Поскольку память выделяется страницами, то в конце секций часто можно найти свободное пространство, использование которого нами никак не повлияет на работоспособность системы.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Этап 7: Создание и внедрение шелл-кода

Задача шелл-кода состоит в выполнении нескольких команд в пространстве пользователя. Так, для запуска установленного приложения можно использовать Activity Manager (am): /system/bin/am start -n com.eltechs.originaldoom/.doomDemo.DoomDemo.

Перед этим шелл-код должен выполнить ряд других важных действий:

- защиту от повторного выполнения;
- сохранение контекста процессора;
- настройку переменных окружения, необходимых для работы Package/Activity Manager;
- вызов оригинального обработчика getpriority.

Шелл-код представляет собой предварительно скомпилированный позиционно независимый машинный код для архитектуры ARM64, в конце которого размещена таблица указателей на нужные элементы, заполняемая на стороне модема.

```
printk_address:
    .quad 0xffffff800810a6c0      // функция printk
call_umh_address:
    .quad 0xffffff80080bfe00      // функция call_usermodehelper
syscall_address:
    .quad 0xffffff80080bbebc      // оригинальная getpriority
argv:
    .quad 0xffffff800915df00      // массив аргументов
envp:
    .quad 0xffffff800915df40      // переменные окружения
marker_address:
    .quad 0xffffff800915dff0      // маркер однократного выполнения
```


© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

В коде также содержится таблица переменных окружения, необходимых для корректного функционирования пользовательских процессов am (Activity Manager), pm (Package Manager) и других.

```
umh_env_path:

    .asciz

"PATH=/sbin:/system/sbin:/product/bin:/apex/com.android.runtime/bin:/system/bin:/sy
stem/xbin:/odm/bin:/vendor/bin:/vendor/xbin"

umh_env_boot_cp:

    .asciz

"BOOTCLASSPATH=/apex/com.android.runtime/javalib/core-
oj.jar:/apex/com.android.runtime/javalib/core-libart.jar:..."
```

Этап 8: Модификация таблицы системных вызовов

Финальным шагом является замена указателя на обработчик системного вызова getpriority в таблице sys_call_table на адрес нашего шелл-кода, предварительно размещенного в свободной области в секции кода ядра. После успешной модификации наш код автоматически выполняется при следующем вызове getpriority. Поскольку этот системный вызов периодически используется разными компонентами Android, активация происходит в течение короткого времени без дополнительного вмешательства.

В конце работы шелл-код восстанавливает состояние регистров процессора и вызывает оригинальный системный вызов getpriority.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

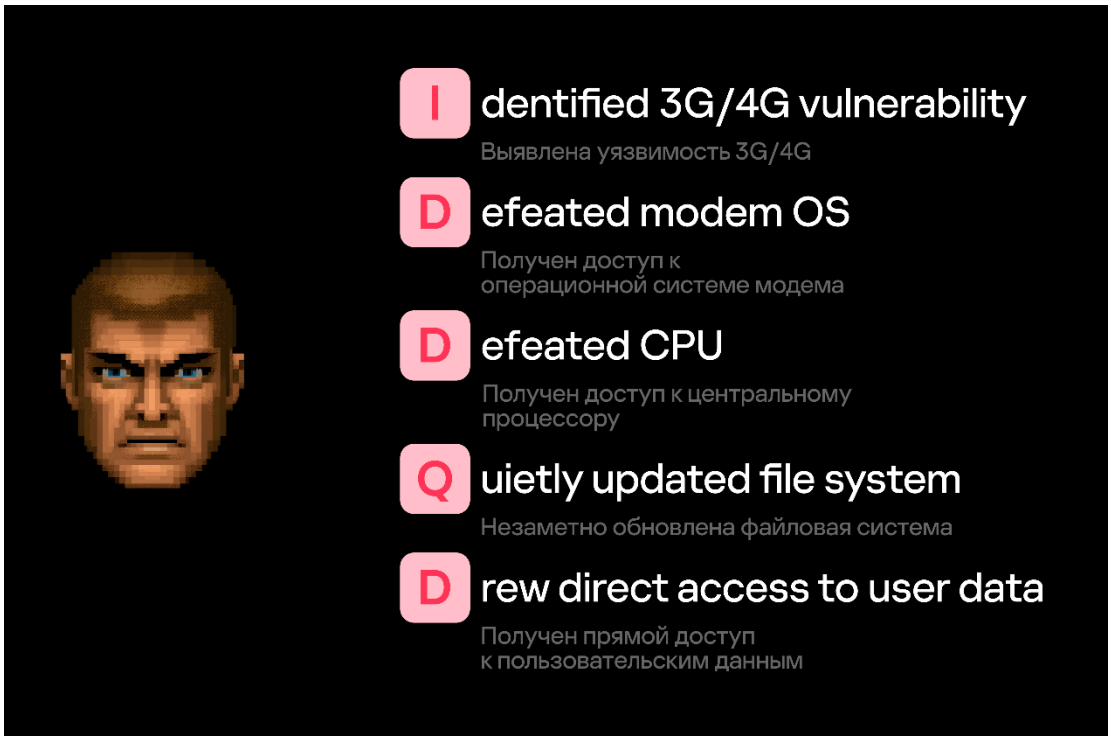
```
    // Восстановление регистров
    ldp x9, x30, [sp], #16
    ldp x6, x8, [sp], #16
    ldp x4, x5, [sp], #16
    ldp x2, x3, [sp], #16
    ldp x0, x1, [sp], #16
call_original:
    // Вызов оригинальной функции
    adr x2, syscall_address
    ldr x2, [x2]
    br x2
```

В результате работы всей цепочки эксплойтов мы получаем возможность исполнить свой код на стороне AP.

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Теперь мы можем с уверенностью сказать, что головные устройства некоторых китайских автомобилей годятся для игры в Doom. При этом весь процесс осуществляется удаленно и без участия пользователя, успешно обходя все существующие механизмы безопасности Android и ядра Linux благодаря прямому доступу к памяти.

Заключение



© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Эксплуатация всего лишь одной уязвимости на стороне модема позволила получить полный контроль (God Mode) над всем SoC. Важно отметить: хотя данная уязвимость может быть исправлена программно, обнаруженные проблемы микроархитектурного уровня возможно исправить только в будущих партиях конкретного SoC. Возникает резонный вопрос: характерна ли эта аппаратная «особенность» только для конкретного типа SoC?

Захватив SoC, злоумышленник получает не только возможность контролировать информационный поток между устройством и внешним миром, но и практически неограниченный доступ к наиболее важным компонентам конечного устройства. При компрометации SoC, используемой в головном устройстве автомобиля, злоумышленник вряд ли ограничится установкой Doom. Он может получить удаленный доступ к пользовательским данным, в частности к записи голоса через встроенный микрофон, или развить атаку на подключаемые мобильные устройства. Более того, в случае ошибки в конфигурации бортового шлюза шины CAN он может получить возможность удаленного воздействия на другие автомобильные блоки[3].

Проблема усугубляется еще и тем, что при обнаружении серьезной уязвимости в модеме потребуется значительное время на обновление всех устройств, использующих данный SoC. При этом в некоторых устройствах функция удаленного обновления может быть не предусмотрена. В таком случае установка обновления потребует дополнительных усилий и затрат со стороны производителя конечного устройства, поскольку каждый уязвимый SoC придется обновлять вручную.

[3] Об угрозах удаленного воздействия на автомобили читайте в материалах «Тренды информационной безопасности современного автомобиля» и «Исследователи безопасности – главный мотиватор автопроизводителей инвестировать в защиту своих продуктов».

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»

Центр реагирования на инциденты информационной безопасности промышленных инфраструктур «Лаборатории Касперского» (Kaspersky ICS CERT) — глобальный проект «Лаборатории Касперского», направленный на координацию усилий производителей систем автоматизации, владельцев и операторов промышленных объектов, а также исследователей ИТ-безопасности для защиты промышленных предприятий от кибератак. Kaspersky ICS CERT направляет свои усилия в первую очередь на выявление потенциальных и существующих угроз, нацеленных на системы промышленной автоматизации и промышленный интернет вещей.

Kaspersky ICS CERT ics-cert@kaspersky.com

© 2025 АО «ЛАБОРАТОРИЯ КАСПЕРСКОГО»