Операция «Триангуляция»

Георгий Кучерин
Леонид Безвершенко
Борис Ларин

Операция «Триангуляция»

Целенаправленная атака на устройства Apple

Метод заражения: цепочка эксплойтов нулевого дня, не требующая взаимодействия с пользователем

Обнаружена Лабораторией Касперского

В Лаборатории Касперского

Программа презентации

История самой изощренной атаки, когда-либо обнаруженной ЛК!

1. **Как мы обнаружили и поймали все этапы атаки**

2. **Обзор цепочки заражения**
   - Удаленное выполнение кода
   - Обход аппаратной подписи указателей (PAC)
   - Уязвимость ядра
   - **Обход аппаратной защиты памяти (PPL)**

3. **Обзор шпионского ПО**

Как мы обнаружили и поймали все этапы?

Злоумышленники допустили ошибки

Ошибки злоумышленников

1. Подозрительная сетевая активность

Подозрительная сетевая активность

Использованные домены:

backuprabbit[.]com

cloudsponcer[.]com

snoweeanalytics[.]com

...

Подозрительные имена

Всегда в парах

Высокий исходящий трафик

Cloudflare + NameCheap

Подозрительная сетевая активность

Перед появлением доменов всегда происходит активность iMessage!

Time	Server Name	Destination	Destination Port	Protocol
222.577175	init.ess.apple.com	62.115.253.208	443	TLSv1.3
223.248546	kt-prod.ess.apple.com	17.145.0.2	443	TLSv1.3
250.471089	p113-caldav.icloud.com	17.250.84.36	443	TLSv1.2
301.339923	edge-102.sesto4.icloud-content.com	17.250.84.37	443	TLSv1.3
302.194211	p31-content.icloud.com	17.250.84.22	443	TLSv1.2
314.766744	setup.icloud.com	17.250.84.19	443	TLSv1.2
339.869951	backuprabbit.com	104.21.21.154	443	TLSv1.3
359.630968	gsa.apple.com	17.32.194.2	443	TLSv1.2
360.605764	backuprabbit.com	104.21.21.154	443	TLSv1.3
361.092903	pds-init.ess.apple.com	62.115.253.218	443	TLSv1.3
368.065719	cloudsponcer.com	104.21.79.172	443	TLSv1.3
377.414078	backuprabbit.com	104.21.21.154	443	TLSv1.3
423.442812	gateway.icloud.com	17.250.84.4	443	TLSv1.3
426.333906	identity.ess.apple.com	17.138.176.4	443	TLSv1.3
427.062256	identity.ess.apple.com	17.138.176.4	443	TLSv1.3
428.386581	identity.ess.apple.com	17.138.176.4	443	TLSv1.3
429.057571	identity.ess.apple.com	17.138.176.4	443	TLSv1.2
437.411511	iphone-ld.apple.com	62.115.253.233	443	TLSv1.3
500.156738	init.itunes.apple.com	184.51.132.49	443	TLSv1.3
760.068442	courier.push.apple.com	17.57.146.133	5223	TLSv1.3
761.825773	iphone-ld.apple.com	62.115.253.219	443	TLSv1.3
762.498958	cloudsponcer.com	104.21.79.172	443	TLSv1.3
765.125499	gs-loc.apple.com	17.36.206.4	443	TLSv1.3

Сервер вложений iMessage

Плохие домены

Ошибки злоумышленников

1. Подозрительная сетевая активность
2. Артефакты заражения

Артефакты заражения

Злоумышленники удаляли вложения с эксплойтами, но забыли про папки => пустые папки в **Library/SMS/Attachments**

Удаляли данные из множества SQL баз, но забыли о **datausage.sqlite** (статистика сетевого трафика)

10:05 Datausage **BackupAgent** (Bundle ID: , ID: 710) WIFI IN: 0.0, WIFI OUT: 0.0 - WWAN IN: 734459.0, WWAN OUT: 287912.0

Сетевая активность из:

a) **Устаревшего приложения**
b) **Которое не использует сеть**

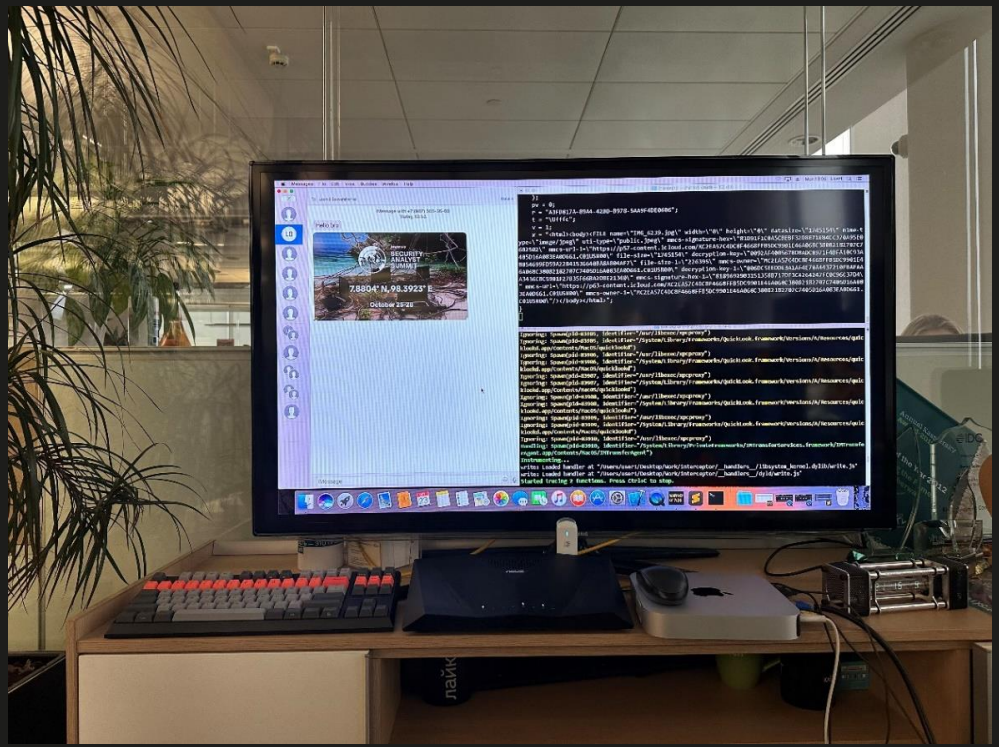
План Перехват: инструментация iMessage на Mac

Авторизация в Apple ID на Mac

Перехват функций iMessage при помощи Frida

Ожидание нового заражения...

Не сработало... 😔



Ошибки злоумышленников

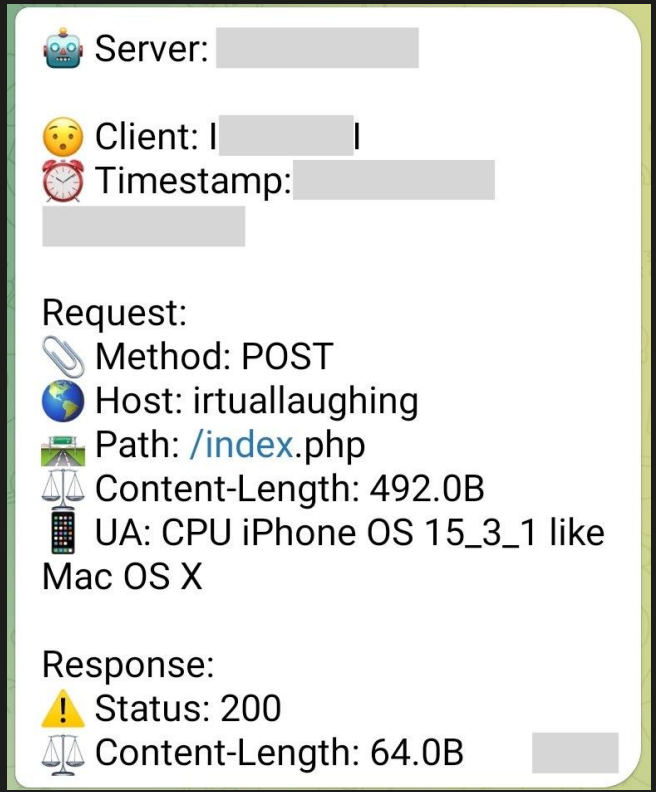
1. Подозрительная сетевая активность
2. Артефакты заражения
3. Отсутствие обнаружения атаки «человек посередине» !

Мы установили ловушку

1. Установили mitmproxy (для перехвата HTTPS) на сервер
2. Установили свой SSL-сертификат на айфоны коллег
3. Направили весь трафик с айфонов на сервер с mitmproxy

Результат

Мы можем наблюдать за атаками!



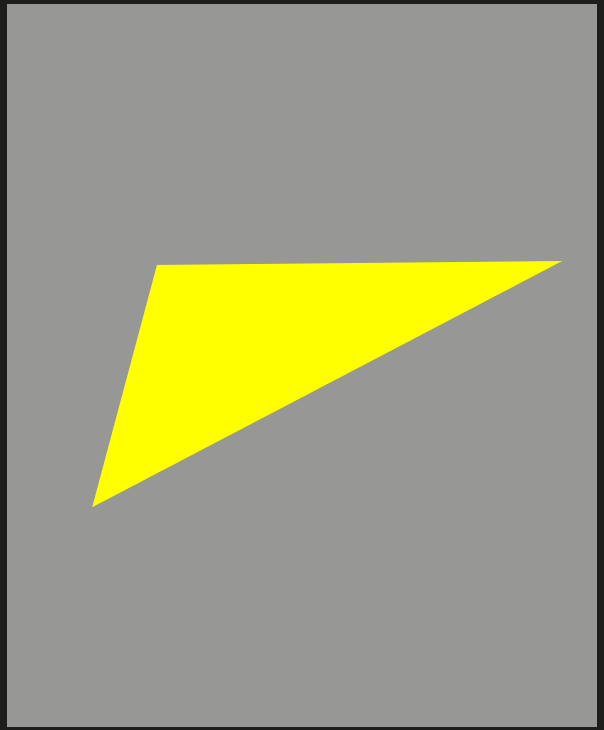
И патчить код атакующих для получения следующих этапов атаки!

Что нас ждало внутри

1. Снятие отпечатка устройства

```
context.bufferData(context.ELEMENT_ARRAY_BUFFER, l,
context.STATIC_DRAW);
context.useProgram(C);
context.clearColor(0.5, 0.7, 0.2, 0.25);
context.clear(context.COLOR_BUFFER_BIT);
context.drawElements(context.TRIANGLES, l.length,
context.UNSIGNED_SHORT, 0);
C.L = context.getAttribLocation(C, Z('VE'));
C.W = context.getUniformLocation(C, Z('Zv'));
context.enableVertexAttribArray(C.L);
context.vertexAttribPointer(C.L, 3, context.FLOAT, !1, 0, 0);
context.uniform2f(C.W, 1, 1);
context.drawArrays(context.TRIANGLE_STRIP, 0, 3);
var h = new Uint8Array(262144);
```

Хэш нарисованного **треугольника** и многое другое



Что нас ждало внутри

2. Слои и слои асимметричной криптографии...

... мы их обошли внедрением своих ключей с помощью **regex**

Стади

NOOOOOOO! YOU CANT JUST SNIFF AND DECRYPT ALL HTTPS TRAFFIC

haha mitmproxy go brrrr

рвер
ующих
я

Что нас ждало внутри

3. Валидаторы, эксплойты, шпионское ПО, плагины

Мы заполучили все компоненты атаки!

Цепочка атаки

Аккаунт iMessage атакующих

PDF файл

Эксплойт TrueType

TrueType

- **Самый распространенный формат шрифтов в мире**
- **Разработан Apple и выпущен в мае 1991 года**
- **Шрифты содержат программы для виртуальной машины**
- **Apple поделилась кодом с Microsoft**
- **Код Apple был добавлен в Windows 3.1**
- **В результате в Windows найдено множество уязвимостей**

Эксплойт написан на языке ассемблера TrueType VM

1. Я прочитал документацию
2. Начал разрабатывать модуль IDA Pro
3. В документации отсутствуют некоторые детали
4. Решил найти их в коде интерпретатора TrueType VM на iOS

TrueType на iOS


```
DCQ fnt_ROLL(fnt_LocalGraphicStateType *)
DCQ fnt_BinaryOperand(fnt_LocalGraphicStateType *)
DCQ fnt_BinaryOperand(fnt_LocalGraphicStateType *)
DCQ fnt_SCANTYPE(fnt_LocalGraphicStateType *)
DCQ fnt_INSTCTRL(fnt_LocalGraphicStateType *)
DCQ fnt_ADJUST(fnt_LocalGraphicStateType *)
DCQ fnt_ADJUST(fnt_LocalGraphicStateType *)
DCQ fnt_GETVARIATION(fnt_LocalGraphicStateType *)
DCQ fnt_GETDATA(fnt_LocalGraphicStateType *)
DCQ fnt_IDefPatch(fnt_LocalGraphicStateType *)
DCQ fnt_IDefPatch(fnt_LocalGraphicStateType *)
DCQ fnt_IDefPatch(fnt_LocalGraphicStateType *)
```

Недокументированная инструкция

Недокументированная инструкция

Я нигде не смог найти информации об этой инструкции...
...за исключением древних исходников Windows OS

```
/* Apple Only Instructions */
/* fnt_ADJUST */
case 0x8f:
/* fnt_ADJUST */
case 0x90:
...
{
 // Error
 break;
}
```

Перевод: **/* Инструкции только для Apple */**

Единственное упоминание и оно из начала 90-х!

Уязвимость #1: CVE-2023-41990

Одна недокументированная инструкция
Предназначенная только для Apple
Сможете догадаться, где уязвимость?

Уязвимость #1: CVE-2023-41990


```
/* Adjust points from start to end */

while(True)
{
  points[current_point] = new_value;
  flags[current_point] |= MOVED;

  if (current_point < end_point)
    current_point++;
  else
    current_point = start_point;

  value = points[current_point];

  if (value >= limit_start && value <= limit_end)
    if (count == 0)
      break;
  else
    break;

  count--;
}
```

end_point

Буфер с “points”

Буфер со смещениями массивов **(!)**

```
/* Adjust points from end to start */

while(True)
{
  points[current_point] = new_value;
  flags[current_point] |= MOVED;

  if (current_point > start_point)
    current_point--;
  else
    current_point = end_point; -1

  value = points[current_point];

  if (value >= limit_start && value <= limit_end)
    if (count == 0)
      break;
  else
    break;

  count--;
}
```

Хронология исправления

Обработчик инструкции fnt_ADJUST присутсвовал с начала 90-х

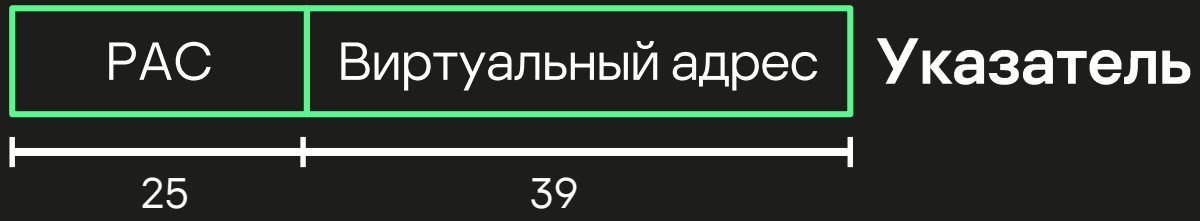
Тихо удален в январе 2023 из iOS 16.3, macOS 13.2

К тому времени уже активно использовался злоумышленниками

iOS 15, macOS 12 исправлены в июле 2023 после нашего оповещения

Обход защиты Pointer Authentication Code (PAC)

В **arm64e** специальные инструкции подписывают и проверяют указатели



Атакующим необходимо подписать указатели с помощью PACIZA

Злоумышленники используют эти недостатки:

1. Слишком много разных указателей подписываются PACIZA
2. Подписанные указатели можно найти в памяти

Уязвимость #2 (исправлена в iOS 16.0)

1. Атакующие находят **подписанный** указатель функции dlsym
2. Используют для получения **подписанного** указателя mprotect
3. Создают поддельный заголовок Mach-O
4. Его экспорт указывает на функцию **signPointer** из **dyld**
5. Для заголовка создают поддельный объект dyld4::Loader
6. Объект инъецируется в список загруженных модулей
7. dlsym возвращает **подписанный** указатель функции signPointer!

signPointer подписывает любой указатель любым ключом/модификатором ключа = полный обход PAC

Цепочка атаки

Аккаунт iMessage атакующих

PDF файл

Эксплойт TrueType

CVE-2023-41990
Обход PAC

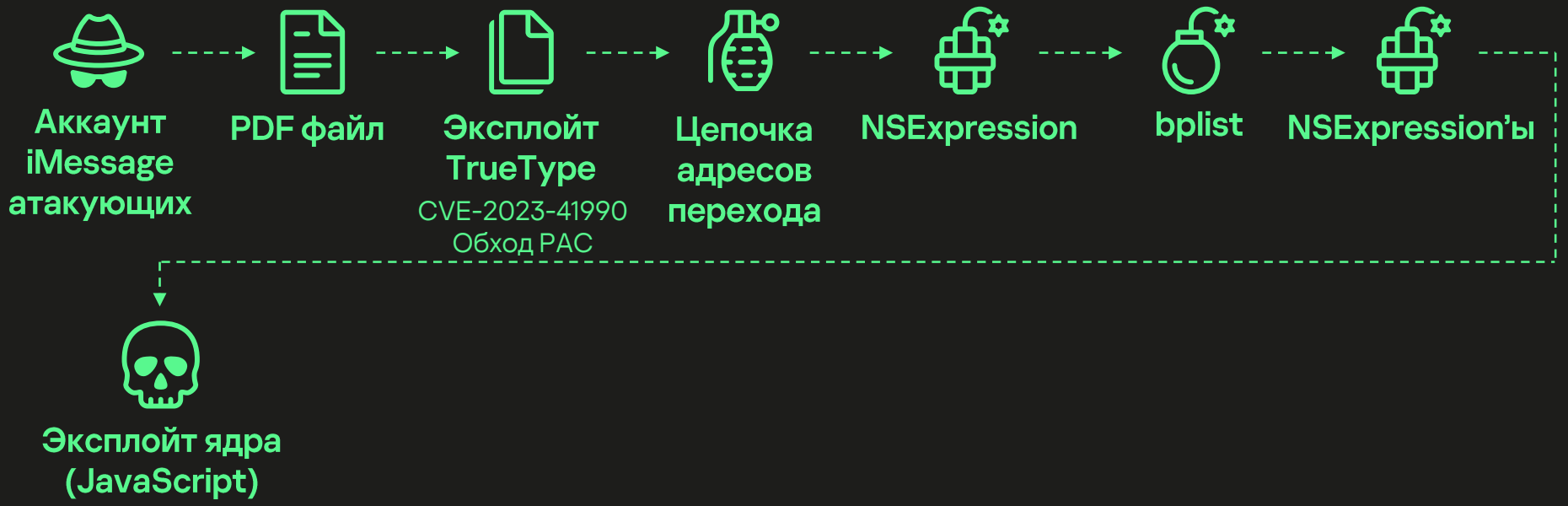
Цепочка адресов перехода

NSExpression

Классы NSExpression и NSPredicate


```
now(FUNCTION(FUNCTION(..., "_copyFormatStringWithConfiguration:" ,
        FUNCTION(CAST("NSNumber", "Class"), "numberWithLong:" ,
            FUNCTION(FUNCTION(CAST("NSValue", "Class"), "valueWithBytes:objCType:" ,
                FUNCTION((FUNCTION(CAST("NSNumber", "Class"), "numberWithLong:" , SELF) +
                    FUNCTION(FUNCTION(CAST("NSString", "Class"), "stringWithUTF8String:" , SELF),
                        "valueForKey:" , "length")) + 1, "unsignedLongValue"),
                FUNCTION("^v", "UTF8String")), "pointerValue"))), ...),
    FUNCTION(FUNCTION(CAST("NSKeyedUnarchiver", "Class"), "unarchiveObjectWithData:" ,
        FUNCTION(FUNCTION(CAST("NSData", "Class"), "dataWithBytes:length:" ,
            FUNCTION((FUNCTION(CAST("NSNumber", "Class"), "numberWithLong:" , SELF)
             + FUNCTION(FUNCTION(CAST("NSString", "Class"), "stringWithUTF8String:" , SELF),
                "valueForKey:" , "length")) + 9, "unsignedLongValue"),
                FUNCTION(..., "longValue")),
                    "decompressedDataUsingAlgorithm:error:" , FUNCTION(2, "longValue"), nil)),
                        "expressionValueWithObject:context:" , nil, nil)) == 1
```

Цепочка атаки



Модификация работы библиотеки JavaScriptCore

Приложения могут использовать JavaScript (без JIT-компиляции)
Злоумышленники извлекают из этого максимальную выгоду

```
WTF::setPermissionsOfConfigPage(void)::onceFlag = -1;
bmalloc::availableMemory(void)::onceFlag = -1;
bmalloc::availableMemory(void)::availableMemory = 0x100000;

JSC::Config::disableFreezingForTesting();
JSC::Config::enableRestrictedOptions();
JSDisableGCTimer();
WTF::ThreadGroup::addCurrentThread(WTF::activeThreads(void)::activeThreads);

putenv("JSC_useMachForExceptions=0");
putenv("JSC_useGC=0");
putenv("JSC_useDollarVM=1");
```

```
>>> describe([1, 2, 3, 4])
Object: 0x1062b4340 with butterfly 0x8000e4008 (Structure 0x106
    ArrayWithInt32, Proto:0x1062c80a0, Leaf]), StructureID: 97
```

Отладочный функционал $vm позволяет манипулировать памятью

Эксплойт JavaScript

Самый сложный эксплойт JavaScript, который я видел

Обфусцирован с помощью UglifyJS

- Код нечитабелен
- Размер кода сведен к минимуму, но...

11000 строк кода!

после переформатирования с помощью jsbeautifier

JavaScript на стероидах

JavaScript

Исполнение произвольного кода / JS на стероидах

Эксплойт выполняет функции ОС напрямую из JavaScript

```
async call_mach_vm_allocate_api(target, address, size, flags) {
    var address_ptr = set_value_wrap(this.set_value_helper, address);
    var result = (await this.api_caller.call_api('mach_vm_allocate', [target, address_ptr, size, flags])).get_dword();
    var address = get_value_wrap(address_ptr);
    return [result, address]
}

async call_mach_vm_protect_api(target, address, size, set_maximum, prot) {
    return (await this.api_caller.call_api('mach_vm_protect', [target, address, size, set_maximum, prot])).get_dword();
}
```

JSCallbackObject + цепочка гаджетов + оператор **instanceof**

Уязвимость #3: CVE-2023-32434

Невероятно простая и мощная уязвимость ядра

Целочисленные переполнения в системных вызовах отображения памяти!

Уязвимость #3: CVE-2023-32434

mach_make_memory_entry - создаёт запись памяти

```
kern_return_t mach_make_memory_entry(
        vm_map_t            target_map,
        vm_size_t           *size,   // входной параметр - размер
        vm_offset_t         offset,    // входной параметр - смещение
        vm_prot_t           permission,
        ipc_port_t          *object_handle,
        ipc_port_t          parent_entry)
```

Уязвимость #3: CVE-2023-32434

Этот системный вызов имел такую проверку смещения и размера!

```
if ((offset + *size + parent_entry->data_offset) > parent_entry->size) {
    kr = KERN_INVALID_ARGUMENT;
    goto make_mem_done;
}
```

Проверка обходится с size = `0xFFFFFFFFFFFFC000` и offset = `0x8000`

Создает vm entry (запись памяти) с размером `0xFFFFFFFFFFFFC000`!

Уязвимость #3: CVE-2023-32434

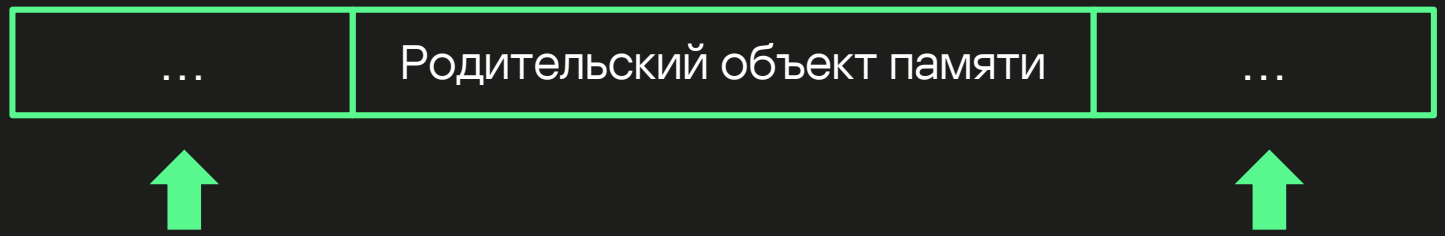
vm_map – отображает объект памяти на область виртуальной памяти

Этот системный вызов имел такую проверку :

```
if (named_entry->size < (offset + initial_size)) {
    return KERN_INVALID_ARGUMENT;
}
```

Проверка **не работает** для size = `0xFFFFFFFFFFFFC000`!

Уязвимость #3: CVE-2023-32434

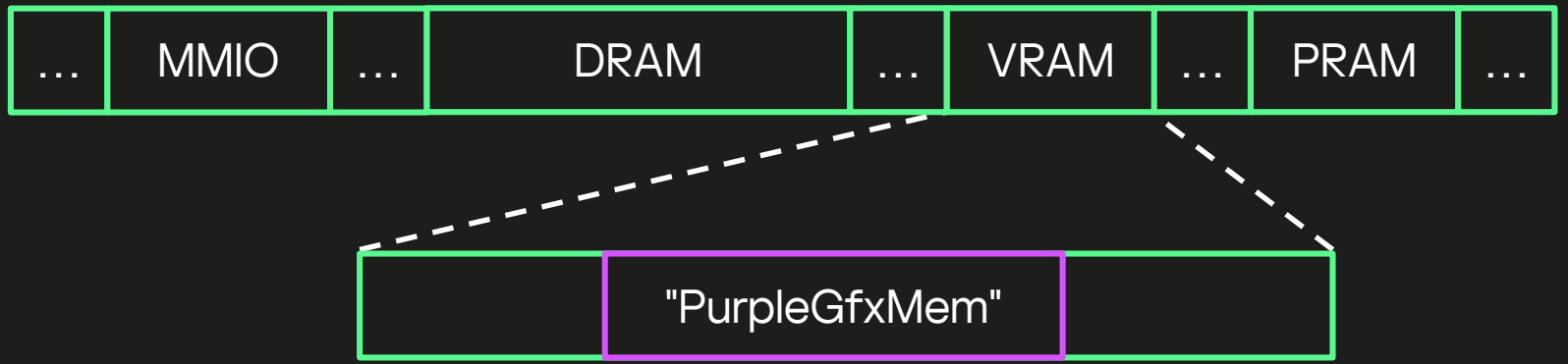


Злоумышленники могут отображать память за пределами родительского объекта памяти!

Использование **особого** родительского объекта памяти позволяет отображать **любую физическую память** на уровне пользователя!

Уязвимость #3: CVE-2023-32434

Карта физической памяти



Кадровый буфер (видеобуфер)

Может использоваться с **mach_make_memory_entry**!

Уязвимость #3: CVE-2023-32434

1. Атакующие создают вредоносную vm entry для PurpleGfxMem
2. Получают базовый адрес PurpleGfxMem
3. Находят iboot-handoff и базовый адрес DRAM
4. Получают диапазон адресов образа ядра из регистров MMIO
5. Находят структуры cpu_data для каждого CPU
6. Находят массив vm_page_array
7. Создают поток и получают его адрес ядра с cpu_active_thread
8. Получают физический адрес потока с vm_page_array
9. Читают память по ядерным адресам функцией thread_get_state

Уязвимость #4: CVE-2023-38606

Жизнь iOS хакера на самом деле ~~несчастна~~... счастлива

1. Удаленное выполнение кода
2. Обход PAC
3. Уязвимость ядра
4. ~~Обход PPL, KTRR, Kernel PAC~~

Злоумышленник

Уязвимость #4: CVE-2023-38606

В чипах Apple A12-A16 существует **особая** аппаратная функция

Которая позволяет обойти аппаратную защиту памяти ядра и записать желаемые данные по защищенному адресу

Для этого просто нужно:

1. Записать желаемый адрес
2. Записать желаемые данные
3. Записать “хэш” данных (???)

В аппаратные регистры MMIO, не используемые прошивкой!

Центральный чип Apple (система на кристалле)

Это не только процессор (CPU), это множество периферийных устройств

CPU
AOP
SMC
SBR
AIC
DWI
SPMI

GPIO
PMS
PCIE
SIO
PWM
I2C
SPI

UART
AES
AUDIO
MCA
DPA
MCC
DCS

USB
JPG
MSR
PMP
APCIE
GFX
+ много еще

CPU работает с ними через отображенные в памяти I/O (MMIO) регистры

Дерево устройств (DeviceTree)

MMIO адреса устройств хранятся в файле дерева устройств

```
reg                         0x00000000
AAPL,phandle                0x00000011
cpu-id                      0x00000000
acc-impl-reg                00 00 f0 10 02 00 00 00  00 00 05 00 00 00 00 00
no-aic-ipi-required
l2-cache-size               0x00400000
function-error_handler      1b 00 00 00 48 72 72 45  00 00 00 00
cpu-uttdbg-reg              00 00 04 10 02 00 00 00  00 00 01 00 00 00 00 00
interrupt-parent            0x0000001c
name                        cpu0
l2-cache-id                 0x00000000
```

Пример: диапазон MMIO адресов устройства **acc-impl** для **cpu0**:
начало=0x210f00000, **размер=0x50000**

Таинственные MMIO адреса используемые эксплойтом


```
0x206040000
0x206140008
0x206140108
0x206150020
0x206150040
0x206150048
```

Я проверил **разные деревья устройств**, для разных устройств и разных прошивок
- ни одного упоминания об этих адресах

Я проверил **исходники, образы ядра**, **модули ядра**, **загрузчик**, **прошивки чипов**
- ни одного упоминания об этих адресах

Записал **обращения к памяти** на Mac с чипами M1/M2 с помощью **m1n1**
- ни одного упоминания об этих адресах

Таинственные MMIO адреса используемые эксплойтом

Откуда злоумышленникам известно, как использовать неиспользуемые MMIO?

Какие периферийные устройства используются для эксплуатации?

Что это может быть?

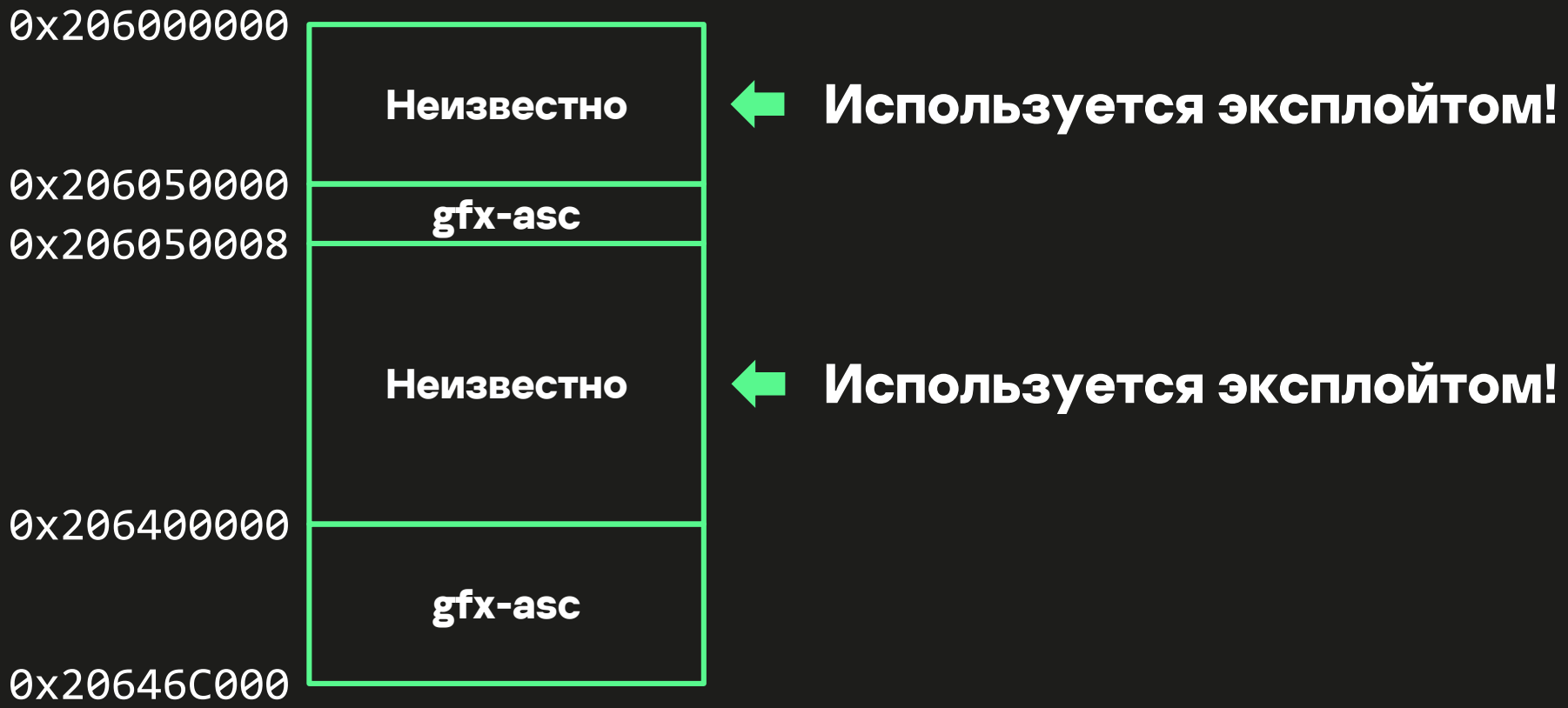
Давайте проверим, какие известные MMIO находятся рядом

```
compatible          iop,ascwrap-v2
iommu-parent        0x000000b1
interrupt-parent    0x0000001c
interrupts          99 01 00 00 98 01 00 00  9b 01 00 00 9a 01 00 00
clock-gates         0x000000d1
clock-ids           0x0000013b
reg                 00 00 40 06 00 00 00 00  00 00 02 00 00 00 00 00
                    00 00 05 06 00 00 00 00  08 00 00 00 00 00 00 00
AAPL,phandle        0x000000af
iop-version         0x00000001
device_type         gfx-asc
role                GFX
power-gates         0x000000d1
name                gfx-asc
```

Сопроцессор графического процессора

[0x206400000-0x20646C000, 0x206050000-0x206050008]

Теперь мы знаем, кому принадлежат эти адреса!



Как используются таинственные MMIO? MMIO #1

0x206040000
0x206140008
0x206140108
0x206150020
0x206150040
0x206150048

Используется только в начале и в конце работы

```
[redacted]():
    value = read_qword(0x206040000)
    if ((value & 0x90000000) != 0):
        return

    write_qword(0x206040000, value | 0x80000000)
    while (True):
        if ((read_qword(0x206040000) & 0x10000000) != 0):
            break

[redacted]():

    value = read_qword(0x206040000)

    value = (value & 0xFFFFFFFF2FFFFFFF) | 0x40000000
    write_qword(0x206040000, value)
    while (True):
        if ((read_qword(0x206040000) & 0x10000000) == 0):
            break
```

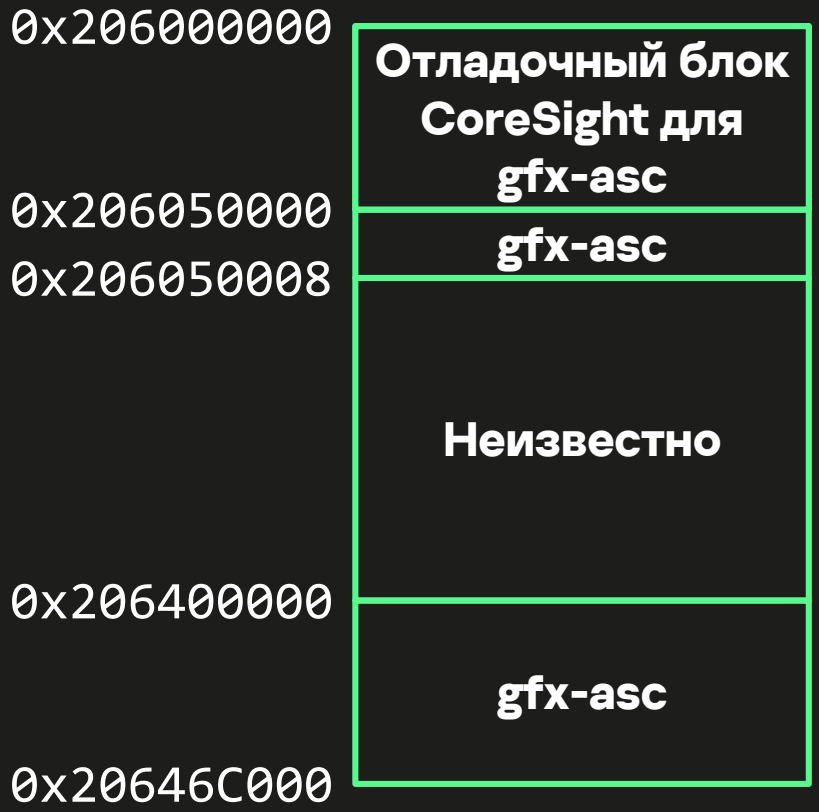
Как используются таинственные MMIO? MMIO #1


```
ml_dbgwrap_halt_cpu():

value = read_qword(0x206040000)
if ((value & 0x90000000) != 0):
    return

write_qword(0x206040000, value | 0x80000000)
while (True):
    if ((read_qword(0x206040000) & 0x10000000) != 0):
        break
```

```
dbgwrap_status_t
ml_dbgwrap_halt_cpu(int cpu_index, uint64_t timeout_ns)
{
    ...

    /* Clear all other writable bits besides dbgHalt;
    none of the power-down or reset bits must be set. */
    *(dbgwrap_reg_t*)(cdp->coresight_base[CORESIGHT_UTT]
        + DBGWRAP_REG_OFFSET) = DBGWRAP_DBGHALT;

    ...

    dbgwrap_reg_t *dbgWrapReg = (dbgwrap_reg_t *)(
        cdp->coresight_base[CORESIGHT_UTT] + DBGWRAP_REG_OFFSET);

    while (!(*dbgWrapReg & DBGWRAP_DBGACK)) {
        ...
    }
}
```

`0x206040000` это отладочный регистр ARM/Apple CoreSight (UTT) для сопроцессора графического процессора!

Мы решили половину головоломки!



Используется эксплойтом!

Используется эксплойтом!

Как используются таинственные MMIO? MMIO #2, 3

0x206040000 → остановка/запуск сопроцессора GPU

0x206140008 →
0x206140108 → остановка/запуск аппаратной функции
0x206150020
0x206150040
0x206150048

Как используются таинственные MMIO? MMIO #4

`0x206040000` → остановка/запуск сопроцессора GPU

`0x206140008` → остановка/запуск аппаратной функции

`0x206140108` → остановка/запуск аппаратной функции

`0x206150020` → только для Apple A15/A16, пишется 1

`0x206150040`

`0x206150048`

Как используются таинственные MMIO? MMIO #5

`0x206040000` →	остановка/запуск сопроцессора GPU
`0x206140008` →	остановка/запуск аппаратной функции
`0x206140108` →	
`0x206150020` →	только для Apple A15/A16, пишется 1
`0x206150040` →	флаги (???) + нижняя часть адреса
`0x206150048`	

Как используются таинственные MMIO? MMIO #6

`0x206040000` →	остановка/запуск сопроцессора GPU
`0x206140008` →	остановка/запуск аппаратной функции
`0x206140108` →	
`0x206150020` →	только для Apple A15/A16, пишется 1
`0x206150040` →	флаги (???) + нижняя часть адреса
`0x206150048` →	данные + “хэш” данных (???) вместе с верхней частью адреса и неизвестной составляющей

Как используются таинственные MMIO?


```
dma_init(original_value_0x206140108)

write_qword(0x206150040, 0x2000000 | (phys_addr & 0x3FC0))

pos = 0
while (pos < 0x40):
    write_qword(0x206150048, read_qword(data + pos))
    pos += 8

hash1 = calculate_hash(data)
hash2 = calculate_hash(data+0x20)

phys_addr_upper = ((((phys_addr >> 14) & mask) << 18) & 0x3FFFFFFFFFFF)
value = phys_addr_upper | (hash1 << i) | (hash2 << 50) | 0x1F
write_qword(0x206150048, value)

dma_done(original_value_0x206140108)
```

Что можно с этим сделать?

- Изменять **записи в таблице страниц**
- Изменять данные в защищенном сегменте **__PPLDATA**
- Изменять код в сегменте **__TEXT_EXEC** *

“Хэш”


```
sbox = [
    0x007, 0x00B, 0x00D, 0x013, 0x00E, 0x015, 0x01F, 0x016,
    0x019, 0x023, 0x02F, 0x037, 0x04F, 0x01A, 0x025, 0x043,
    0x03B, 0x057, 0x08F, 0x01C, 0x026, 0x029, 0x03D, 0x045,
    0x05B, 0x083, 0x097, 0x03E, 0x05D, 0x09B, 0x067, 0x117,
    0x02A, 0x031, 0x046, 0x049, 0x085, 0x103, 0x05E, 0x09D,
    0x06B, 0x0A7, 0x11B, 0x217, 0x09E, 0x06D, 0x0AB, 0x0C7,
    0x127, 0x02C, 0x032, 0x04A, 0x051, 0x086, 0x089, 0x105,
    0x203, 0x06E, 0x0AD, 0x12B, 0x147, 0x227, 0x034, 0x04C,
    0x052, 0x076, 0x08A, 0x091, 0x0AE, 0x106, 0x109, 0x0D3,
    0x12D, 0x205, 0x22B, 0x247, 0x07A, 0x0D5, 0x153, 0x22D,
    0x038, 0x054, 0x08C, 0x092, 0x061, 0x10A, 0x111, 0x206,
    0x209, 0x07C, 0x0BA, 0x0D6, 0x155, 0x193, 0x253, 0x28B,
    0x307, 0x0BC, 0x0DA, 0x156, 0x255, 0x293, 0x30B, 0x058,
    0x094, 0x062, 0x10C, 0x112, 0x0A1, 0x20A, 0x211, 0x0DC,
    0x196, 0x199, 0x256, 0x165, 0x259, 0x263, 0x30D, 0x313,
    0x098, 0x064, 0x114, 0x0A2, 0x15C, 0x0EA, 0x20C, 0x0C1,
    0x121, 0x212, 0x166, 0x19A, 0x299, 0x265, 0x2A3, 0x315,
    0x0EC, 0x1A6, 0x29A, 0x266, 0x1A9, 0x269, 0x319, 0x2C3,
    0x323, 0x068, 0x0A4, 0x118, 0x0C2, 0x122, 0x214, 0x141,
    0x221, 0x0F4, 0x16C, 0x1AA, 0x2A9, 0x325, 0x343, 0x0F8,
    0x174, 0x1AC, 0x2AA, 0x326, 0x329, 0x345, 0x383, 0x070,
    0x0A8, 0x0C4, 0x124, 0x218, 0x142, 0x222, 0x181, 0x241,
    0x178, 0x2AC, 0x32A, 0x2D1, 0x0B0, 0x0C8, 0x128, 0x144,
    0x1B8, 0x224, 0x1D4, 0x182, 0x242, 0x2D2, 0x32C, 0x281,
    0x351, 0x389, 0x1D8, 0x2D4, 0x352, 0x38A, 0x391, 0x0D0,
    0x130, 0x148, 0x228, 0x184, 0x244, 0x282, 0x301, 0x1E4,
    0x2D8, 0x354, 0x38C, 0x392, 0x1E8, 0x2E4, 0x358, 0x394,
    0x362, 0x3A1, 0x150, 0x230, 0x188, 0x248, 0x284, 0x302,
    0x1F0, 0x2E8, 0x364, 0x398, 0x3A2, 0x0E0, 0x190, 0x250,
    0x2F0, 0x288, 0x368, 0x304, 0x3A4, 0x370, 0x3A8, 0x3C4,
    0x160, 0x290, 0x308, 0x3B0, 0x3C8, 0x3D0, 0x1A0, 0x260,
    0x310, 0x1C0, 0x2A0, 0x3E0, 0x2C0, 0x320, 0x340, 0x380
]
```

Свой собственный алгоритм?

```
def calculate_hash(buffer):

    acc = 0
    for i in range(8):
        pos = i * 4
        value = read_dword(buffer + pos)
        for j in range(32):
            if (((value >> j) & 1) != 0):
                acc ^= sbox[32 * i + j]

    return acc
```

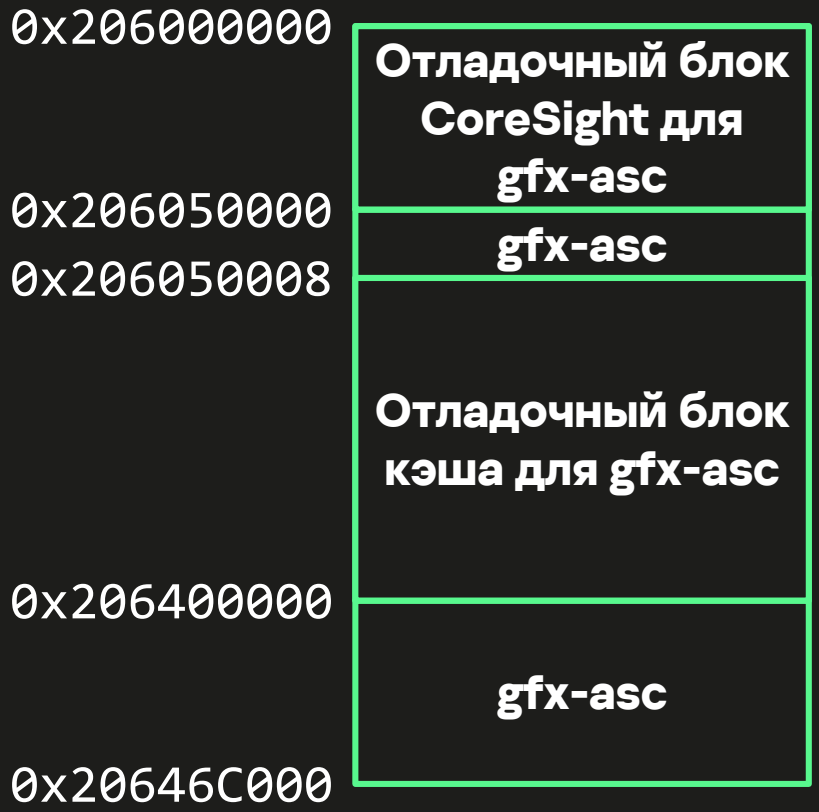
Мы нигде не смогли найти эту таблицу

“Хэш”

Оказалось что “хэш” это код коррекции ошибок (ECC)

Точнее, Код Хэмминга с уникальной таблицей поиска

Мы решили головоломку?



Используется эксплойтом!

Используется эксплойтом!

Это не обычная уязвимость

Как они узнали, как использовать эту неизвестную аппаратную функцию?

Что нужно было знать злоумышленникам

- Что функция отладки для кэша существует
- Что для ее работы необходим Код Хэмминга
- Значения в уникальной таблице поиска
- Расположение и назначение всех регистров MMIO
- Как и в каком порядке с ними взаимодействовать

Исправление Apple

pmap-io-ranges в дереве устройств содержит таблицу с диапазонами запрещенных адресов

```
0x620000000     0x20000000      0x27    'PCIe'
0x640000000     0x40000000      0x27    'PCIe'
0x2412C0000         0x4000    0x4007    'DART'
...
0x24A80C000         0x4000    0x4007    'DAPF'
...
0x267020000         0x4000    0x4007    'SMMU'
...
0x601008000         0x4000    0x4007    'DSID'
```

“Peripheral Component Interconnect Express”

“Device Address Resolution Table”

“Device Address Filter”

“System Memory Management Unit”

...

```
0x206000000          0x50000    0x4007    'DENY'
0x206110000         0x2F0000    0x4007    'DENY'
```

Перевод: **“Отказ”**

Цепочка атаки

Валидатор

Запускает действия из скрипта:

- “DeleteLogs” (Удалить логи)
- “DeleteArtifacts” (Удалить артефакты)
- “ProcessList” (Список процессов)
- “InterfaceList” (Список интерфейсов)
- “JailbreakDetect” (Обнаружение джейлбрейка)
- “EnableAdTracking” (Включить отслеживание рекламы)
- “DeviceInfo” (Информация об устройстве)
- “InstalledApps” (Установленные приложения)

Неиспользованное:

- “PSPDetect” (???)

Уборка

Удаляет логи сбоев (crash logs) для этих процессов:

imagent MessagesBlastDoorService IMTranscoderAgent	Следы цепочки эксплойтов iMessage
com.apple.WebKit.WebContent MobileSafari	Следы цепочки эксплойтов WebKit
BackupAgent nehelper	Следы вредоносного ПО

Атакующие сами раскрыли с каких аккаунтов велись атаки!

40 MD5 хэшей Apple ID !

099882568724cd63474797e41566041a	mailto:travislong544@yahoo.com
1b69a6070fe6e464dd56acc7765c3264	mailto:norsarall87@outlook.com
2bcb505a88f37b5199b7efd2b5ad8aca	mailto:jesteristhebestband@gmail.com
3aa73cf8508bd31fd49755fe96a918b8	mailto:homicidalwombat@yahoo.com
3b1111fff15d32f5a972c88c84722eb2	mailto:nigelmlevy@gmail.com
3d527800ad9418b025340775eaf6454c	mailto:supercatman15@hotmail.com
54dbb2f4760be4f7de4b56bb11a33d4c	mailto:shannonkelly404@gmail.com
57a6461482442c11f34ae9e41701325e	mailto:superhugger21@gmail.com
596d06451664010f5c2b0ceff5cf5a00	mailto:parkourdiva@yahoo.com
5cdcd68fa020521a7efe558e963a8cd4	mailto:naturelover1972@outlook.com

Мы взломали эти хэши

Обнаружение джейлбрейка и инструментов

Известные

- TaiG
- Pangu
- Chimera
- Electra
- Unc0ver
- Evasion7
- Checkra1n

+ 30 универсальных индикаторов

- /Applications/Cydia.app
- /private/var/tmp/cydia.log
- /var/lib/cydia
- /Library/Frameworks/CydiaSubstrate.framework
- /Library/MobileSubstrate
- /usr/sbin/frida-server
- ....

+ Обнаружение виртуальных устройств Corellium

Валидатор

Запускает действия из скрипта:

- “DeleteLogs” (Удалить логи)
- “DeleteArtifacts” (Удалить артефакты)
- “ProcessList” (Список процессов)
- “InterfaceList” (Список интерфейсов)
- “JailbreakDetect” (Обнаружение джейлбрейка)
- “EnableAdTracking” (Включить отслеживание рекламы)
- “DeviceInfo” (Информация об устройстве)
- “InstalledApps” (Установленные приложения)

Неиспользованное:

- “PSPDetect” (???)

Зачем нужно проверять наличие PSP на iOS?

PSPDetect

(PSP) personal security product

definition

продукт личной безопасности

Definition of *personal security product* : noun

1. Software applications installed on end-user workstations designed to protect users from Internet and network threats like exploit, Trojans, port scans, or viruses.

Продукты личной безопасности существуют для macOS

Разведка

Эти данные используются для проверки того, была ли заражена **правильная** жертва:

- Емкость батареи
- Имя пользователя (полное)
- Имя пользователя (краткое)
- Идентификатор рекламы
- Связанные устройства
- Номер телефона
- Номер устройства
- Имя устройства
- iMessage токены
- UID
- Серийный номер
- Время работы
- Активен ли VPN
- Apple ID
- IMEI

Цепочка атаки

Шпионское ПО

Полноценная APT-платформа

- Код написан на Objective-C
- Получает и запускает дополнительные плагины
- Коммуникационный стек: Protobuf и HTTPS
- В коде есть признаки того, что он существовал **10+ лет**

Какая информация собирается?

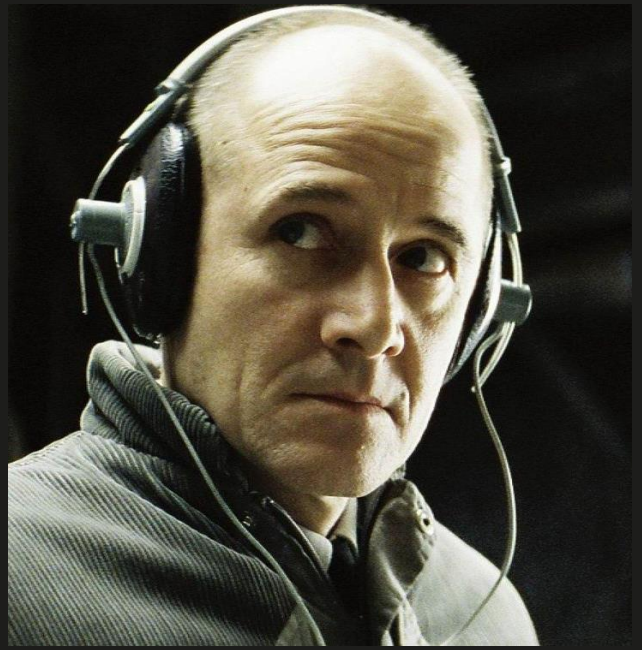
Собирается почти всё

- Геолокация (местоположение, скорость, направление)
- Полный доступ ко всем файлам и базам данных
- Извлечение ключей из связки ключей iOS
- Мессенджеры (WhatsApp, iMessage, Telegram)
- Доступ к камере и микрофону
- Доступ к адресной книге
- Информация о других устройствах Apple поблизости (?)

Прослушка

Свой формат: Аудио + метаданные

Нет записи при включенном экране



Фотографии

Очень сложно обрабатывать все фотографии

Как они справляются?

Искусственный интеллект и машинное обучение

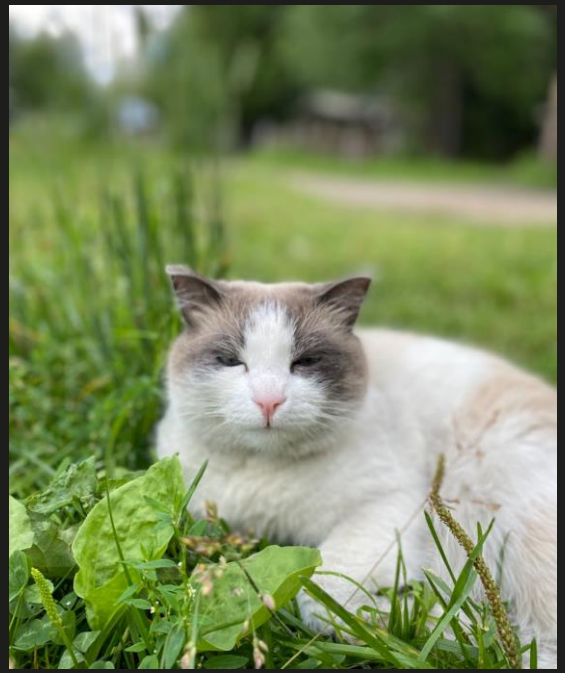
Что видит человек:

Что видит iOS/macOS:

```
Орех          Кокос          Еда
Вода          Купальник      Океан
Природа       Песок          Пляж
Напиток с трубочкой

Найден текст: BEACH
Найден текст: NANA
```

Искусственный интеллект и машинное обучение

Что видит человек:

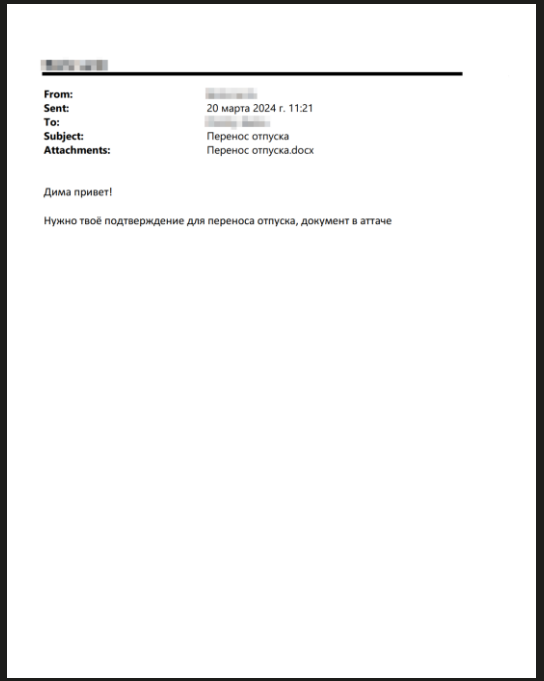


Что видит iOS/macOS:

- Листва
- Растение
- Животное
- Кошка
- Природа
- Млекопитающее
- Трава
- Земля

Искусственный интеллект и машинное обучение

Что видит человек:



From:
Sent: 20 марта 2024 г. 11:21
To:
Subject: Перенос отпуска
Attachments: Перенос отпуска.docx

Дима привет!

Нужно твоё подтверждение для переноса отпуска, документ в аттаче

Что видит iOS/macOS:

Распечатанная страница

Документ

Выводы

Собирайте (и проверяйте) сетевые логи на работе и дома!

Аппаратные средства защиты бесполезны, пока существуют другие аппаратные функции, позволяющие их обойти!

Благодаря машинному обучению вредоносы будут знать все!

kaspersky

Спасибо!

kaspersky