iOS 9.3.3 越狱漏洞分析
&
iOS 10 安全机制改进

Team Pangu

议程

- CVE-2016-4654
- 漏洞利用分析
- iOS 10安全机制
- iPhone 7安全机制
- 结论

内核漏洞时间线

- MOSEC 2016上我们展示了iOS 10 beta1的越狱视频
- 该漏洞在iOS 10 beta2中被修复
- 于是我们在2016.7.24发布了针对 iOS 9.2-9.3.3的越狱
  - 通过安装一个应用直接攻击内核
- 苹果在2016.8.4上午突然推送iOS 9.3.4来修补该漏洞
  - https://support.apple.com/en-us/HT207026
- 我们当天下午在Blackhat 2016发表演讲

CVE-2016-4654

- 应用内可以直接触发该漏洞
- 该漏洞是IOMobileFrameBuffer的一个堆溢出
  - 溢出的长度任意可控
  - 溢出的数据内容部分可控
- Jonathan Levin 在他的 “*OS Internals” 第三卷中会对我们的越狱进行详细的逆向分析

CVE-2016-4654

- “IOMobileFramebuffer::swap_submit(IOMFBSwap *)”
  - IOMFBSwap数据可控（用户态传入）
  - v33等于v31
  - v31从swap+216+4*v15处读取
  - 循环中没有检查v33长度
  - 赋值v34的时候发生堆溢出

```
v28 = swap + 4 * v15;
v30 = request + 4 * v15;
*(_DWORD *)(v30 + 176) = *(_DWORD *)(v28 + 176) & 7;
*(_QWORD *)(request + 304) = *(_QWORD *)swap;
*(_QWORD *)(request + 312) = *(_QWORD *)(swap + 8);
*(_QWORD *)(request + 320) = *(_QWORD *)(swap + 16);
v31 = *(_DWORD *)(v28 + 216);
*(_DWORD *)(v30 + 380) = v31;
if ( v31 )
{
  v32 = 0;
  v33 = (unsigned int *)(v30 + 380);
  v34 = (_OWORD *)(request + (v15 << 6) + 392);
  v35 = (__int128 *)v16;
  do
  {
    v36 = *v35;
    ++v35;
    *v34 = v36;
    ++v34;
    ++v32;
  }
  while ( v32 < *v33 );
}
```

IOMobileFrameBuffer介绍

- 处理屏幕帧缓存的内核扩展
- 用户态通过IOMobileFramebuffer.framework框架来控制
- 观察iPhone 6设备的ioreg输出
  - AppleMobileADBE0 <class IORegistryEntry:IOService:IOMobileFramebuffer:AppleDisplayPipe:AppleH7DisplayPipe:AppleCLCD:AppleMobileADBE0, id 0x1000001de, registered, matched, active, busy 0 (4 ms), retain 9>
- 通过IOServiceOpen打开IOMobileFramebufferUserClient
  - IOServiceMatching匹配“AppleCLCD”

IOMobileFrameBuffer介绍

- 通过externalMethod定位sMethods表

```
sMethods        IOExternalMethodDispatch <sub_FFFFFF801B145D88, 3, 0, 0, 0>
                                         ; DATA XREF: __text:FFFFFF801B144350↑o
                                         ; IOMobileFrameBufferUserClient_start+2C↑o
                IOExternalMethodDispatch <sub_FFFFFF801B145DA8, 0, 0, 0, 0>
                IOExternalMethodDispatch <sub_FFFFFF801B145DA8, 0, 0, 0, 0>
                IOExternalMethodDispatch <sub_FFFFFF801B145DCC, 2, 0, 1, 0>
                IOExternalMethodDispatch <IOMobileFrameBufferUserClient__swap_begin, \
                                          0, 0, 1, 0>
                IOExternalMethodDispatch <IOMobileFrameBufferUserClient__swap_submit, \
                                          0, 0xFFFFFFFF, 0, 0>
                IOExternalMethodDispatch <sub_FFFFFF801B145EAC, 3, 0, 0, 0>
```

- selector=5调用swap_submit，传入一个结构体
  - 最终会调用IOMobileFramebuffer::swap_submit触发漏洞
- selector=4调用swap_begin，返回一个整型
  - 创建一个IOMFBSwapIORequest对象，该对象需要传递给swap_submit
  - 返回的整型是request对象的id

swap_submit

- 用户态传入IOMFBSwap结构的数据
  - 检查结构体的大小必须是544 (9.3.x) 或者424 (9.2.x)
- 首先通过保存在swap+24处的id获取之前创建的IOMFBSwapIORequest对象
- 然后通过传入的swap数据初始化request对象，处理过程在一个循环体中（索引0-2）
  - 处理过程会通过保存在swap+28/32/36处的id查找对应的IOSurface对象，并将对象指针保存到request+32/36/40处
  - 将swap+228数据填充到request+392的时候发生堆溢出
    - 保存在swap+216/220/224的长度没有检查
- swap_submit退出前会检查数据是否合法，如果失败的话会去释放IOMFBSwapIORequest和IOSurface对象

议程

- CVE-2016-4654
- 漏洞利用分析
- iOS 10安全机制
- iPhone 7安全机制
- 结论

控制溢出

- 溢出的长度完全可控（从input+216处读取）
- IOMFBSwapIORequest对象的大小是872，被分配在kalloc.1024 zone
  - 可以覆盖kalloc.1024 zone中紧邻对象的数据
- 堆溢出发生在从input+228拷贝数据到request+392的时候
  - 由于对输入的IOMFBSwap结构有长度检查，无法直接控制覆盖的数据内容
  - 实际上输入的结构体会被包裹在一个mach消息结构中，而该消息结构体同样是在kalloc.1024 zone
  - 通过堆风水可以控制未初始化区域的内存数据

下一步?

- 控制kalloc.1024 zone的内存布局
  - [IOMFBSwapIORequest]+[被覆盖对象]
- 修改被覆盖对象后可以获得代码执行机会
- 首先需要绕过KASLR
- 如何选择被覆盖的对象?

漏洞利用思路A

- 寻找kalloc.1024 zone中的某个对象，并且在对象开始保存了大小
- 通过堆溢出增加对象大小
- 释放到错误的zone -> 读取/修改kalloc.1024中紧邻的对象
  - 不适用于iOS 10（稍后讨论）
  - 稳定性稍差（kalloc.1024 zone中仅有4个对象在同一页面）
  - 该利用思路适用于32位和64位设备

漏洞利用思路B

- 目标为iOS 10 beta + 64位设备
  - SMAP保护并不存在，内核中可以访问用户态地址
- 选择覆盖IOMFBSwapIORequest对象
  - request+16保存了下一个request对象的地址，形成一个链表
  - request+0保存了vtable地址
  - request+328保存了request的id
  - 修改下一个request地址到一个用户态地址从而劫持整个链表
    - 可以读取/修改用户态下伪造的IOMFBSwapIORequest

泄露内核地址

- 再次调用swap_submit，传入伪造的request id以及有效的IOSurface id
  - 能够从request+32读取IOSurface对象的指针
- 可以获取“IOMFB Debug Info”数据信息，会返回当前IOMFB的一些状态信息
  - 包含了所有的swap request的信息
    - 获取request信息的同时会尝试返回IOSurface的数据

泄露内核地址

- 获取数据时会从IOSurface+12读取4个字节作为“src_buffer_id”

```
setDictionaryNumber(dict, (__int64)"src_buffer_Id", *(unsigned int *)(iosurface + 12), 32LL);
if ( *(_DWORD *)(iosurface + 176) )
{
  v9 = (*(__int64 (__fastcall **)(__int64, _QWORD))(*(_QWORD *)iosurface + 224LL))(iosurface, 0LL);
  v10 = "src_stride";
  v11 = v9;
}
else
{
  v11 = *(unsigned int *)(iosurface + 152);
  v10 = "src_stride";
}
```

- 修改request+32处的IOSurface地址为IOSurface-12
  - 获取IOSurface vtable的低4位地址
- 再次修改为IOSurface-8读取vtable的高4位地址
- 从而可以计算出内核的实际加载地址

内核代码执行

- 如果传入的swap数据不合法，swap_submit退出前会调用 IOMFBSwapIORequest::release

```
CBZ             X0, loc_FFFFFF801B14C1DC
LDR             X8, [X0] ; X0=IOMFBSwapIORequest
LDR             X8, [X8,#0x28]
BLR             X8
B               loc_FFFFFF801B14C1DC
```

- 伪造的request对象的vtable完全可控
- X0和X8的数据内容都可控

任意内核读

- 选择Gadgets

```
LDR             X8, [X0]
LDR             X2, [X8,#0xA8]
LDR             X1, [X0,#0x40] ; Control X1
BR              X2
```

```
LDR             X9, [X1,#0x78]
LDR             W9, [X9,#0x18] ; read 4 bytes
STR             W9, [X0,#0x50]
MOV             X0, X8
RET
```

任意内核写

- 选择Gadgets

```
LDR             X8, [X0]
LDR             X2, [X8,#0xA8]
LDR             X1, [X0,#0x40] ; Control X1
BR              X2
```

```
LDR             X8, [X8,#0x688]
ADD             X8, X8, X0
STR             X8, [X1] ; write 8 bytes
RET
```

漏洞修补

```
v32 = *( DWORD *)(v29 + 216);
if ( v32 > 4 )
  v32 = 4;
*((_DWORD *)v30 + v16 + 94) = v32;
if ( v32 )
{
  v33 = 0LL;
  v34 = v69;
  v35 = (unsigned int *)(v69 + 4 * v16 + 376);
  v36 = v17;
  do
  {
    *(_OWORD *)((char *)v30 + v36 + 160) = *(_OWORD *)((char *)v2 + v36);
    ++v33;
    v36 += 16LL;
    v30 = (_QWORD *)v34;
  }
  while ( v33 < *v35 );
  v30 = (_QWORD *)v34;
}
```

议程

- CVE-2016-4654
- 漏洞利用分析
- iOS 10安全机制
- iPhone 7安全机制
- 结论

JIT内存保护

- 支持--X属性的映射页面
- 为JIT物理内存创建两个映射
  - 一个属性是--X
  - 一个属性是RW-
  - 丢弃RW-映射的地址

内核堆管理理

- iOS 9

- 并不是所有的zone都有page meta数据

- 释放到错误的zone仍然有效（如果目标zone没有page meta数据）

  - 能够绕过KASLR并获取代码执行

内核堆管理理

- iOS 10
  - 所有的zone都有page meta数据
  - 释放到错误的zone会被zfree函数检测

```
struct zone_page_metadata *page_meta = get_zone_page_metadata((struct zone_free_element *)addr, FALSE);
if (zone != PAGE_METADATA_GET_ZONE(page_meta)) {
    panic("Element %p from zone %s caught being freed to wrong zone %s\n", addr, PAGE_METADATA_GET_ZONE(page_meta)->zone_name, zone->zone_name);
}
```

内核堆管理理

- 新的kfree_addr函数会根据要释放的堆地址自动获堆的大小
- 修改对象的大小不再有效

```
vm_size_t
kfree_addr(
    void        *addr)
{
    vm_map_t        map;
    vm_size_t       size = 0;
    kern_return_t   ret;
    zone_t          z;

    size = zone_element_size(addr, &z);
    if (size) {
        zfree(z, addr);
        return size;
    }
```

沙盒加强

- Platform规则限制更严格
  - 二进制的规则大小从0x10DE (9.3)变成0x1849 (iOS 10)
  - iOS 10中对更多行为进行了限制
    - file-map-executable
    - system-kext-query
    - process-exec-interpreter
    - process-exec*
    - file-write-create
    - …

KPP

- kernelcache内存布局变化
- 所有的代码和常量放在一起
- 所有的RW数据放在一起
- KPP处理更简单
- __got段被KPP所保护！

com.apple.driver.AppleD2333PMU:__got	FFFFFFF006FFEF00	FFFFFFF006FFF280
com.apple.driver.AppleD2333PMU:__mod_init_func	FFFFFFF006FFF280	FFFFFFF006FFF298
com.apple.driver.AppleD2333PMU:__mod_term_func	FFFFFFF006FFF298	FFFFFFF006FFF2B0
com.apple.driver.AppleD2333PMU:__const	FFFFFFF006FFF2B0	FFFFFFF0070009F0
com.apple.driver.AppleD2333PMU:GAP_hidden	FFFFFFF0070009F0	FFFFFFF007004000
__TEXT:HEADER	FFFFFFF007004000	FFFFFFF007007CE0
__TEXT:__const	FFFFFFF007007CE0	FFFFFFF00701F698
__TEXT:__cstring	FFFFFFF00701F698	FFFFFFF00705E9AA
__TEXT:__os_log	FFFFFFF00705E9AA	FFFFFFF00705FFFF
__DATA_CONST:__mod_init_func	FFFFFFF007060000	FFFFFFF007060210
__DATA_CONST:__mod_term_func	FFFFFFF007060210	FFFFFFF007060418
__DATA_CONST:__const	FFFFFFF007064000	FFFFFFF0070BFBE8
__TEXT_EXEC:__text	FFFFFFF0070C0000	FFFFFFF00753EC88
__KLD:__text	FFFFFFF007540000	FFFFFFF0075416DC
__KLD:__cstring	FFFFFFF0075416DC	FFFFFFF007541EA8
__KLD:__const	FFFFFFF007541EA8	FFFFFFF007541F10
__KLD:__mod_init_func	FFFFFFF007541F10	FFFFFFF007541F18
__KLD:__mod_term_func	FFFFFFF007541F18	FFFFFFF007541F20
__KLD:__bss	FFFFFFF007541F20	FFFFFFF007541F21
__LAST:__pinst	FFFFFFF007544000	FFFFFFF007544020
__LAST:__mod_init_func	FFFFFFF007544020	FFFFFFF007544028
__DATA:__data	FFFFFFF007548000	FFFFFFF007578CC8
__DATA:__sysctl_set	FFFFFFF007578CC8	FFFFFFF00757ADE0
__DATA:__bss	FFFFFFF00757B000	FFFFFFF0075F5828
__DATA:__common	FFFFFFF0075F6000	FFFFFFF0075F7130
com.apple.iokit.IONetworkingFamily:__data	FFFFFFF007658000	FFFFFFF0076580C8
com.apple.iokit.IONetworkingFamily:__common	FFFFFFF0076580C8	FFFFFFF007658430
com.apple.iokit.IONetworkingFamily:__bss	FFFFFFF007658430	FFFFFFF0076584B8
com.apple.iokit.IONetworkingFamily:GAP_hidden	FFFFFFF0076584B8	FFFFFFF0076584C0

KPP

- 时间窗口攻击仍然有效
  - 在短时间内修改/还原代码
- 内核堆可以被标记为RWX
  - 存放内核shellcode
- iPhone 7下有所不同？

AMFI

- 修复了validateCodeDirectoryHashInDaemon中一个条件竞争的漏洞

- 内核解析可执行文件后会请求用户态的amfid来验证该文件的签名，此时可以替换成合法签名的可执行文件

- 修改后amfid会同时返回验证成功的签名的cdhash，该值必须与当前内核加载的文件的cdhash相同

```
if ( isok == 1 )
{
  if ( (unsigned int)amfi_memcmp(cdhash, &return_cdhash, 20) )
  {
    amfi_IOLog("%s: Possible race detected. Rejecting.\n", v31, v51, v52, v53, v54, &v71);
    isok = 0;
    v70 = 0;
  }
```

AMFI

- iOS 10之前amfid仅检查MISValidateSignature的返回值
  - 重定向MISValidateSignature到返回0的函数即可绕过
- iOS 10会调用MISValidateSignatureAndCopyInfo获取cdhash并返回给内核

```
v24 = MISValidateSignatureAndCopyInfo(v19, v21, &v37);
if ( (_DWORD)v24 )
{
  memcpy(&v39, "<unknown>", 0x100uLL);
  v25 = (void *)MISCopyErrorStringForErrorCode(v24);
  if ( v25 )
  {
    CFStringGetCString(v25, &v39, 0x100uLL);
    CFRelease(v25);
  }
  if ( !*a8 )
    syslog(3, "%s not valid: 0x%x: %s", v15, v24, &v39);
  goto LABEL_19;
}
if ( v37 && (v27 = CFGetTypeID(v37), v27 == CFDictionaryGetTypeID()) )
{
  v28 = (void *)CFDictionaryGetValue(v37, *(_QWORD *)kMISValidationInfoCdHash_ptr);
  if ( v28 )
  {
    v30 = CFGetTypeID(v28);
    if ( v30 == CFDataGetTypeID() )
    {
      *a8 = 1;
      CFDataGetBytes(v28, 0LL, 20LL, cdhash);
```

修复了大量未公开漏洞

- 苹果内部的安全团队也在做漏洞挖掘
  - iOS 10修复了我们的两个漏洞
    - 一个堆溢出和一个UAF
- 安全研究人员向苹果报告漏洞
  - task_t相关的问题
    - https://googleprojectzero.blogspot.jp/2016/10/taskt-considered-harmful.html
  - mach_ports_register问题
    - https://bugs.chromium.org/p/project-zero/issues/detail?id=882
  - …
- 你的漏洞被修复了吗?

议程

- CVE-2016-4654
- 漏洞利用分析
- iOS 10安全机制
- iPhone 7安全机制
- 结论

已知的弱点

- 没有SMAP的保护使得针对64位设备的内核漏洞利用反而更容易
- 目前的KPP设计架构无法阻止时间窗口的攻击
- 内核shellcode允许实现内核级别的rootkit

旧设备的KPP

- 内核运行在EL1
- KPP监控层运行在EL3
- SMC(secure monitor call) 触发异常陷入EL3
- 内核完成初始化后会调用SMC指令告诉KPP监控代码为要保护的内存计算checksum

iPhone 7设备

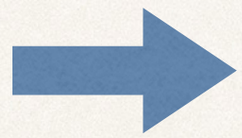
```
BL              _read_random
LDR             X8, [X20]
ORR             X8, X8, #1
STR             X8, [X20]
MOV             W8, #0xC
STR             W8, [SP,#0xE0+var_90]
MOV             W1, #1
MOV             W0, #1
ADD             X2, SP, #0xE0+var_88
ADD             X3, SP, #0xE0+var_90
BL              loc_FFFFFFF0070B3150
LDR             W8, [SP,#0xE0+var_88]
CMP             W8, #3
B.GT            loc_FFFFFFF0070D2D84
ADRP            X8, #byte_FFFFFFF00758D384@PAGE
STRB            W19, [X8,#byte_FFFFFFF00758D384@PAGEOFF]

D2D84                   ; CODE XREF: sub_FFFFFFF0070D0864+2
MOV             W0, #0x801
MOV             X1, #0
MOV             X2, #0
MOV             X3, #0
BL              smc_17
BL              loc_FFFFFFF007365130
STP             XZR,
STR             WZR,
STR             WZR,
STR             XZR,
STR             WZR,
ADRP            X23,
LDR             X20,
LDR             X8, [
MRS             X9, #
CMP             X8, X9
B.EQ            loc_FFFFFFF0070D2DE0
MOV             X0, X20

; =============== S U B R O U T I N E =

smc_17

                SMC             #0x11
                RET
; End of function smc_17
```



```
BL              _read_random
LDR             X8, [X19]
ORR             X8, X8, #1
STR             X8, [X19]
MOV             W8, #0xC
STR             W8, [SP,#0xE0+var_90]
MOV             W1, #1
MOV             W0, #1
ADD             X2, SP, #0xE0+var_88
ADD             X3, SP, #0xE0+var_90
BL              loc_FFFFFFF0070F06CC
LDR             W8, [SP,#0xE0+var_88]
CMP             W8, #3
B.GT            loc_FFFFFFF00711034C
ADRP            X8, #byte_FFFFFFF0075C9384@PAGE
STRB            W23, [X8,#byte_FFFFFFF0075C9384@PAGEOFF]

11034C                  ; CODE XREF: kernel_init+2624↓j
MOV             W0, #0
BL              _ml_set_interrupts_enabled
MOV             X20, X0
LDR             X8, [X22,#qword_FFFFFFF0075CF740@PAGEOFF]
LDR             WZR, [X8,#0x7EC]
MRS             X8, #4, c15, c2, #2
ADRP            X19, #dword_FFFFFFF007004000@PAGE
ADD             X19, X19, #dword_FFFFFFF007004000@PAGEOFF
ADR             X1, a__prelink_text ; "__PRELINK_TEXT"
NOP
ADD             X2, SP, #0xE0+var_88
MOV             X0, X19
BL              sub_FFFFFFF00749E65C
BL              sub_FFFFFFF0071C3CD0
MOV             X21, X0
ADR             X1, a__last ; "__LAST"
NOP
ADD             X2, SP, #0xE0+var_90
MOV             X0, X19
BL              sub_FFFFFFF00749E65C
BL              sub_FFFFFFF0071C3CD0
SUB             X8, X0, #1
LDR             W9, [X24,#0x98]
LSL             W9, W23, W9
NEG             W10, W9
SBFM            X10, X10, #0, #0x1F
AND             X26, X10, X8
LDR             X8, [X28,#0x80]
SUB             X8, X21, X8
LDR             X10, [X27,#0x88]
ADD             X0, X8, X10
LDR             W8, [SP,#0xE0+var_90]
SUB             W8, W8, W21
ADD             W8, W8, W26
ADD             W8, W9, W8
SUB             W1, W8, #1
BL              sub_FFFFFFF0070C230C
LDR             X8, [X22,#qword_FFFFFFF0075CF740@PAGEOFF]
STR             W23, [X8,#0x7EC]
ISB
MSR             #4, c15, c2, #3, X21
MSR             #4, c15, c2, #4, X26
MSR             #4, c15, c2, #2, X23
ISB
BL              sub_FFFFFFF0070CC730
MOV             X0, X20
BL              _ml_set_interrupts_enabled
```

iPhone 7设备的KPP

- 不再有SMC指令的调用
- 初始化代码首先获取了“__PRELINK_TEXT”段和“__LAST”段的物理内存地址，然后将地址存入2个特殊的系统寄存器，该寄存器必须要EL2以上才能访问
- “__PRELINK_TEXT”和“__LAST”之间保存了所有的代码和常量数据
- 新的保护机制通过硬件实现

iPhone 7设备的KPP

- 阻止修改受保护的物理内存
  - 内核代码段无法被修改
  - 时间窗口攻击无效
- 阻止在受保护的物理内存外执行代码
  - 无法运行内核shellcode
  - ROP仍然有效

iPhone 7设备的SMAP

- 同时iPhone 7设备也引入了SMAP的保护
- 访问一个有效的用户态地址会挂起CPU，访问不会得到返回
- 访问一个非法的用户态地址仍然会导致panic

议程

- CVE-2016-4654
- 漏洞利用分析
- iOS 10安全机制
- iPhone 7安全机制
- 结论

结论

- 苹果一直在不断努力改进产品的安全性
- 苹果能通过软硬件结合带来更好的安全保护机制
- iOS内核漏洞的利用越来越困难，也更有价值

Q&A