Analysis of iOS 9.3.3 Jailbreak & Security Enhancements of iOS 10

Team Pangu
Agenda

- CVE-2016-4654
- Exploit Strategy
- iOS 10 Security Enhancements
- iPhone 7 New Protection
- Conclusion
Timeline of the Kernel Bug

- We showed Jailbreak for iOS 10 beta1 on MOSEC 2016
- The bug was fixed in iOS 10 beta2
- We released Jailbreak for 9.2-9.3.3 on 24th July
  - Exploited the kernel bug from an installed App
- Apple published 9.3.4 to fix it on 4th Aug Morning
- We gave a talk at Blackhat 2016 on the same day
CVE-2016-4654

- Any App can exploit this bug to attack kernel
- It’s a heap overflow bug in IOMobileFrameBuffer
- Length of the overflow is controlled
- Data of the overflow is partially controlled
- Full discussion of this, and other past exploits can be found in “*OS Internals” volume III, by Jonathan Levin
CVE-2016-4654

“`IOMobileFramebuffer::swap_submit(IOMFBSwap *)`”

- IOMFBSwap is input from user-land
- v33 comes from v31
- v31 comes from swap+216+4*v15
- No size check of v33 in the loop
- Overflow of v34

```c
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 = (QWORD *)(request + (v15 << 6) + 392);
    v35 = (_int128 *)(v16);
    do
    {
        v36 = *v35;
        ++v35;
        *v34 = v36;
        ++v34;
        ++v32;
    }
    while ( v32 < *v33 );
}
```
Basics of IOMobileFrameBuffer

- It is a kernel extension for managing the screen frame buffer
- It is controlled by the user-land framework IOMobileFramebuffer.framework
- Output from ioreg for iPhone 6
  - AppleMobileADBE0 <class IORegistryEntry:IOService:IOMobileFramebuffer:AppleDisplayPipe:AppleH7DisplayPipe:AppleCLCD:AppleMobileADBE0, id 0x1000001de, registered, matched, active, busy 0 (4 ms), retain 9>
- Open IOMobileFramebufferUserClient via IOServiceOpen
  - IOServiceMatching with “AppleCLCD”
Basics of IOMobileFrameBuffer

- Locate the sMethods table used by externalMethod

<table>
<thead>
<tr>
<th>sMethods</th>
<th>IOExternalMethodDispatch &lt;sub_FFFFFF801B145D88, 3, 0, 0, 0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DATA XRFF: __text::FFFF801B14435010</td>
</tr>
<tr>
<td></td>
<td>IOMobileFramebufferUserClient_swap+2C10</td>
</tr>
</tbody>
</table>
|                | IOExternalMethodDispatch <sub_FFFFFF801B145D88, 0, 0, 0, 0>
|                | IOExternalMethodDispatch <sub_FFFFFF801B145D88, 0, 0, 0, 0>
|                | IOExternalMethodDispatch <sub_FFFFFF801B145DCC, 2, 0, 1, 0>
|                | IOMobileFramebufferUserClient_swap_begin, \                |
|                | 0, 0, 1, 0                                                  |
|                | IOMobileFramebufferUserClient_swap_submit, \               |
|                | 0, 0xFFFFF0, 0, 0                                           |
|                | IOExternalMethodDispatch <sub_FFFFFF801B145EAC, 3, 0, 0, 0>

- selector=5 with input structure is calling swap_submit

- It finally goes to IOMobileFramebuffer::swap_submit to trigger the overflow

- selector=4 with one output scalar is calling swap_begin

- It creates an IOMFBSwapIORequest object which is required for calling swap_submit

- It returns the request id in the output scalar
The input structure is passed to swap_submit as IOMFBSwap data

Size of structure must be 544 for 9.3.x or 424 for 9.2.x

It firstly gets the IOMFBSwapIORequest object by id stored in swap+24

Then it fills the request object according to our input swap in a loop with index from 0 to 2

It will try to find IOSurface by id stored in swap+28/32/36 and save the pointers in request+32/36/40 object

Heap overflow occurs when filling request+392 with swap+228

No size check of count stored in swap+216/220/224

Before exit it will check if the swap is ok, if not it will release IOMFBSwapIORequest and IOSurface objects
Agenda

- CVE-2016-4654
- Exploit Strategy
- iOS 10 Security Enhancements
- iPhone 7 New Protection
- Conclusion
Control the Overflow

- The overflow size is quite easy to control from input+216

- IOMFBSwapIORequest size is 872 in kalloc.1024
  - We can overwrite content of next kalloc.1024 object

- The overflow occurs while copying from input+228 to request+392
  - Remember there is size verification of input so we can’t control the overflow data directly
  - Actually the input data is in a mach message handled by MIG and it’s also in kalloc.1024 zone
  - It’s possible to control the uninitialized memory content by heap fengshui
Next Step?

- Do heap fengshui in kalloc.1024
- `[IOMFBSwapIORequest]+[victim object]`
- We can overwrite data of the victim object
- Need to bypass KASLR
- How to choose the victim object?
Exploit Strategy A

- Find an object in kalloc.1024 and it stores its size at the beginning
- Overwrite the size of the object to a bigger one
- Free into wrong zone -> read/write of next kalloc.1024 kernel object
- Doesn’t work on iOS 10 (we will discuss it later)
- Not so stable because of only 4 objects are in one page for kalloc.1024
- Should work for both 32bit and 64bit devices
Exploit Strategy B

- Target iOS 10 beta + 64bit devices
  - SMAP actually doesn’t exist, kernel mode can access user-land data

- Choose IOMFBSwapIORequest as the victim object
  - All requests are linked, request+16 stores next request pointer
  - request+0 stores vtable pointer
  - request+328 stores the request id

- Overwrite the next pointer to a user-land address to hijack the whole request list
  - We can read/write our controlled fake IOMFBSwapIORequest
Leak Kernel Address

- We call `swap_submit` again with our fake request id and a valid IOSurface id
  - We can get the IOSurface pointer at request+32
- Get property of “IOMFB Debug Info” will give us more detailed informations
  - It will retrieve information of all swap requests
  - Also it will try to get data of IOSurface
Leak Kernel Address

- It will read 4 bytes at IOSurface+12 as “src_buffer_id”

```c
setDictionaryNumber(dict, (__int64)"src_buffer_Id", *(unsigned int *)(iosurface + 12), 32LL);
if ( *(DWORD *)(iosurface + 176) )
{
    v9 = *((__int64 (__fastcall **)(__int64, __QWORD))(*(QWORD *)iosurface + 224LL))(iosurface, OLL);
    v10 = "src_stride";
    v11 = v9;
}
else
{
    v11 = *(unsigned int *)(iosurface + 152);
    v10 = "src_stride";
}
```

- We can set request+32 from IOSurface to IOSurface-12

- Get the lower 4 bytes of IOSurface vtable

- Set it to IOSurface-8 again to get the higher 4 bytes of IOSurface vtable

- We can now calculate the kernel base address
Kernel Code Execution

- Remember if the swap data is not correct, it will call IOMFBSwapIORequest::release before exit

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CBZ</td>
<td>X0, loc_FFFFFF801B14C1DC</td>
</tr>
<tr>
<td>LDR</td>
<td>X8, [X0] ; X0=IOMFBSwapIORequest</td>
</tr>
<tr>
<td>LDR</td>
<td>X8, [X8,#0x28]</td>
</tr>
<tr>
<td>BLR</td>
<td>X8</td>
</tr>
<tr>
<td>B</td>
<td>loc_FFFFFF801B14C1DC</td>
</tr>
</tbody>
</table>

- And we could totally control the vtable of the fake request in user-land memory

- X0 and X8 are under control
Arbitrary Kernel Reading

- Gadgets for reading

```
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
```
Arbitrary Kernel Writing

- Gadgets for writing

```assembly
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
```
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
{
    *((DWORD *)(char *)v30 + v36 + 160) = *((DWORD *)(char *)v2 + v36);
    ++v33;
v36 += 16LL;
v30 = (_QWORD *)v34;
} while ( v33 < *v35 );
v30 = (_QWORD *)v34;
Agenda

- CVE-2016-4654
- Exploit Strategy
- iOS 10 Security Enhancements
- iPhone 7 New Protection
- Conclusion
Hardened JIT Mapping

- --X mapping is now supported
- Create two mappings of the physical JIT memory
  - One is --X
  - One is RW-
- Keeps the location of RW- secret
Kernel Heap Management

- For iOS 9
  - Not all zones has page meta data
  - Free into wrong zone works well when target is none page list zone
    - Enough to bypass KASLR and get code execution
Kernel Heap Management

- For iOS 10
  - There are page meta data for all zones
  - Prevent freeing into wrong zone, check zfree code

```c
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);
}
```
Kernel Heap Management

- New function kfree_addr will automatically get size according to address
- Overwrite size of object no longer works

```c
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 );
        zfree( z, addr );
    }
    return size;
}
```
Enhanced Sandbox

- Platform profile is more restricted
  - Profile size is 0x10DE for 9.3 and 0x1849 for iOS 10
  - More operations are checked of iOS 10
    - file-map-executable
    - system-kext-query
    - process-exec-interpreter
    - process-exec*
    - file-write-create
    - ...

Change of the kernelcache memory layout

Put all code and const together

Put all RW data together

Makes KPP more efficient

__got is now under protection!
KPP

- Time attacking is still practical
- Patch/Restore in a short time window
- Kernel heap can be marked as RWX
- Kernel shell code works well
- BUT different story for iPhone 7!
Fix a potential race in validateCodeDirectoryHashInDaemon

It’s possible to replace the executable file to a valid one after kernel resolve the code signature and ask amfid to verify it

Now amfid will also return the cdhash of the file it verified, the hash must match the one kernel already read

```c
if ( isok == 1 )
{
    if ( (unsigned int)amfi_memcmp(cdhash, &return_cdhash, 20) )
    {
        amfi_IOLog("$: Possible race detected. Rejecting.
", v31, v51, v52, v53, v54, &v71);
        isok = 0;
        v70 = 0;
    }
}```
Before iOS 10 amfid only checks return value of MISValidateSignature

- Easy to bypass by hijacking it to some function just return 0

Now it calls MISValidateSignatureAndCopyInfo instead and get cdhash to return to kernel
Fix Lots of Unpublished Bugs

- Apple security team are hunting bugs
  - Two bugs of ours were fixed in iOS 10
    - One heap overflow and one UAF
- Researchers report bugs to Apple
  - task_t related issues
    - https://googleprojectzero.blogspot.jp/2016/10/taskt-considered-harmful.html
  - Multiple memory safety issues in mach_ports_register
    - https://bugs.chromium.org/p/project-zero/issues/detail?id=882
    - ...
- Did your bugs get patched?
Agenda

- CVE-2016-4654
- Exploit Strategy
- iOS 10 Security Enhancements
- iPhone 7 New Protection
- Conclusion
Known Weakness

- It’s actually easier to write kernel exploit for 64bit devices because of NO SMAP
- Current KPP architecture is not capable to prevent time attacking
- Kernel shellcode allows kernel level rootkit
KPP of Old Devices

- Kernel runs at EL1
- KPP monitor runs at EL3
- SMC (secure monitor call) causes an exception to EL3
- After kernel finish initialization, it calls SMC to tell KPP to init all checksums of protected memory
KPP of iPhone 7

- Apparently there is no SMC

- The initialize code retrieves physical addresses of "__PRELINK_TEXT" and "__LAST" segments. It then store them in special system registers which requires minimum EL=EL2

- All code and const values are between "__PRELINK_TEXT" and "__LAST"

- This new protection is obviously implemented in hardware
KPP of iPhone 7

- It prevents writing to the protected physical memory
  - Can’t touch code memory
  - Time attacking doesn’t work anymore
- It prevents executing outside of the protected physical memory range
  - Can’t execute shellcode in kernel
- ROP is still an option
SMAP on iPhone 7

- Also we notice there is kind of SMAP on iPhone 7
- Dereference valid user-land address will simply hang the CPU, never get return
- Dereference invalid user-land address still cause a panic
Agenda

- CVE-2016-4654
- Exploit Strategy
- iOS 10 Security Enhancements
- iPhone 7 New Protection
- Conclusion
Conclusion

- Apple keeps putting lots of efforts to make their products more secure
- It’s more easier for Apple to bring security feature which is combined with hardware and software
- iOS kernel exploit is now harder and more valuable
Q&A