Recreating an iOS 0-day jailbreak out of Apple’s security patches

August, 2019
Who Am I?

Stefan Esser

- working in IT Security since 1998
- started with runtime encryption / decryption
- moved on to linux daemon security
- then did a lot of work in PHP and Web Application Security
- finally moved on in 2010 to iOS and Mac Security
Motivation

- the release of iOS 12.1.4 fixed vulnerabilities used in the wild
- we only know because of a tweet from someone at Google PO
- neither Apple nor Google officially described the incident
- no official description of the vulnerabilities from those who found it
- 3rd parties had to use patch diffing to figure it out

- Of course Google Project 0 dumped a lot of info literally YESTERDAY :P
Agenda

• Introduction
• Diffing the kernel for CVE-2019-7287
• Diffing user land for CVE-2019-7286
• Exploitation roadmap for CVE-2019-7287
• Exploitation roadmap for CVE-2019-7286
• Conclusion
Introduction
Security Content of iOS 12.1.4

- Apple released iOS 12.1.4 on February 7th 2019
- in response to a serious FaceTime vulnerability
- vulnerability in FaceTime got a lot of media attention
- initiator of a group call could force recipient to answer
Security Content of iOS 12.1.4 (II)

• ... but security updated contained two more vulnerability fixes [1]

**Foundation**

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: An application may be able to gain elevated privileges

Description: A memory corruption issue was addressed with improved input validation.


**IOKit**

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: A memory corruption issue was addressed with improved input validation.

And then Google Project 0 chimed in

- one day after release a tweet from @benhawkes of Google’s P0 appeared [2]
- apparently Google had knowledge of both flaws being exploited in the wild

Ben Hawkes
@benhawkes

CVE-2019-7286 and CVE-2019-7287 in the iOS advisory today (support.apple.com/en-us/HT209520) were exploited in the wild as 0day.

About the security content of iOS 12.1.4
This document describes the security content of iOS 12.1.4.
support.apple.com
Any more info?

- Aside from Ben Hawkes’ tweet there was no further information
- This raised a lot of questions among researchers like
  - How did Google find an iOS 0-day in the wild
  - Who was attacked? Google itself? Or their customers?
  - Was the attack hidden in some app? (because both vulnerabilities seems to be LPE/sandbox escape only)
  - Why did Google keep silent considering iOS 0-days are not often found
  - Will we ever get samples?

GOOGLE dumped a full description yesterday[3]
Any more info? (II)

- most of these questions are still unanswered
- only bit of more info was given within a tweet of another Google researcher
- apparently the two vulnerabilities were combined with an already fixed browser bug
- furthermore it was said that Google were simply too busy to do a write up
- tweet was lost in ocean of twitter

GOOGLE dumped a full description yesterday[3]
References

• [1] About Security Content of iOS 12.1.4
  https://support.apple.com/en-sg/HT209520

• [2] Tweet by @benhawkes
  https://twitter.com/benhawkes/status/1093581737924259840

• [3] A very deep dive into iOS Exploit chains found in the wild
  https://googleprojectzero.blogspot.com/2019/08/a-very-deep-dive-into-ios-exploit.html

• [4] In-the-wild iOS Exploit Chain 4
Diffing the kernel for CVE-2019-7287
CVE-2019-7287

- Information released by Apple is short and misleading

**IOKit**

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: A memory corruption issue was addressed with improved input validation.


- we only learn that it is a memory corruption
- affected component is said to be IOKit
- IOKit is a part of the iOS kernel that is responsible for device drivers (however further analysis will reveal that this is misleading)
Preparing the Kernels

- In order to analyse Apple’s patch we first need to grab a copy of the two kernels
- download IPSW files for iOS 12.1.3 and 12.1.4 via links from TheiPhoneWiki [1]
- unzip the kernelcache files
- use Xerub’s IMG4LIB [2] to extract the macho kernels
- load both kernels into different IDA sessions
- use Lumina [3] and other scripts to fill the databases with symbols
- perform as many steps as possible to cleanup the disassemblies
Diffing the Kernels

• with both kernels being prepared and symbolised we can now use Diaphora [4]
• diffing will take a long while and is best performed overnight
• to speed up diffing we could limit diff to only affected area
• problem is we don’t really know what is affected
• Apple listed IOKit as affected but limiting to main kernel did not get us results
Diffing the Kernels (II)

• when you diff whole kernel you run into several problems
  • lots of unclean disassembly
  • many more functions
  • many more false matches
• so diffing result will be incomplete and harder to interpret
Finding the affected Component (I)

- Diaphora result was hard to read
- However after a while we found a candidate that could be the culprit [5]
Finding the affected Component (II)

- Diaphora found us a newly inserted size check in method ucGetEncryptedSeedSegment of object called ProvInfoIOKitUserClient.
- This could be why Apple mentioned the affected area as IOKit.
- However, this code is not in the main kernel but in driver ProvInfoIOKit.
Analysing ProvInfoIOKit

• Purpose of ProvInfoIOKit was unknown
• Google search for this string revealed no additional information
• driver not accessible from container sandbox or mobile safari
• only sandbox profiles allowing access are
  • findmydevice
  • mobileactivation
  • identityservice
Analysing ProvInfoIOKit (II)

- object **ProvInfoIOKitUserClient** is user space interface of driver
- further research revealed that affected method is an external method of driver
- In total 6 external methods are supported by that object
  
  - **ucGenerateSeed** (obfuscated name: fpXqy2dxjQo7)
  - **ucGenerateInFieldSeed** (obfuscated name: afpHseTGo8s)
  - **ucExchangeWithHoover** (obfuscated name: AEWpRs)
  - **ucGetEncryptedSeedSegment**
  - **ucEncryptSUInfo**
  - **ucEncryptWithWrapperKey**
The Surprise !!! (I)

- Check of other external methods revealed two more new size checks
The Surprise !!! (II)

- Check of other external methods revealed two more new size checks

```
_int64 __fastcall ProvInfoIOKitUserClient::ucEncryptWithWrappedKey(_int64 a1, unsigned int *a2, size_t a3, _
{  _DWORD *v8; // x19
   unsigned int *v9; // x20
   _int64 v10; // x21
   size_t v11; // x22
   _int64 v12; // x19
   char *v13; // x0
   _int64 v15; // x0
   v8 = (_DWORD *)a3;
   v9 = a2;
   v10 = a1;
   if ( !a2 || !a3 )
   {  
      v12 = 0x800002C2LL;
      v13 = "{ProvInfoIOKitUserClient::ucEncryptWithWrappedKey] bad pointer for input or output structure\n"
      goto LABEL_7;
   }
   v11 = a2[17];
   if ( (unsigned int)v11 > 0x40 || a2[34] >= 0x41 )
   {  
      v12 = 0x800002C2LL;
      v13 = "{ProvInfoIOKitUserClient::ucEncryptWithWrappedKey] bad input structure length\n"
      goto LABEL_7;
   }
   v12 = 0x800002C2LL;
   v13 = "{ProvInfoIOKitUserClient::ucEncryptWithWrappedKey] bad input structure length\n"
   return v12;
   memmove(v8 + 1, a2 + 1, v11);
   v8[17] = v9[17];
   memmove(v8 + 18, v9 + 18, v9[34]);
   v8[34] = v9[34];
   v15 = *(*int64 (__fastcall **)(_QWORD, _QWORD, _DWORD *, _QWORD, _DWORD *))(*(_QWORD **)(&v10 + 216) + 137
   *(*_QWORD *)(&v10 + 216),
```
Controlled memmove length

- two methods `ucEncryptSUInfo` and `ucEncryptWithWrapperKey` are better targets
- newly introduced size check is just before size is used in a memmove
- the size for memmove is taken out of the incoming mach message
- and an arbitrary memory block is moved into the out message

```swift
v11 = a2[17];
if ((unsigned int)v11 > 0x40 || a2[34] >= 0x41 ) {
    v12 = 0xE00002C2LL;
    v13 = "[ProvInfoI0KitUserClient::ucEncryptWithWrappedKey] bad input structure length\n";
    LABEL 7:
    ILog(v13);
    return v12;
}
memmove(v8 + 1, a2 + 1, v11);
v8[17] = v9[17];
memmove(v8 + 18, v9 + 18, v9[34]);
v8[34] = v9[34];
```
Controlled memmove length

- two methods **ucEncryptSUInfo** and **ucEncryptWithWrapperKey** are better targets
- newly introduced size check is just before size is used in a memmove
- the size for memmove is taken out of the incoming mach message
- and an arbitrary memory block is moved into the out message
References

• [1] TheiPhoneWiki Firmware Page
  https://www.theiphonewiki.com/wiki/Firmware/iPhone/12.x

• [2] Xerub’s IMG4LIB
  https://github.com/xerub/img4lib

• [3] Hexray’s iOS Kernel Lumina Support
  https://www.hex-rays.com/products/ida/7.2/the_mac_rundown/the_mac_rundown.shtml

• [4] Diaphora
  https://github.com/joxeankoret/diaphora

Diffing user land for CVE-2019-7286
ZecOps Writeups

• a company called ZecOps performed similar research on the user land part
• they released a write up about it [1]
• the following section uses some content from their write up
CVE-2019-7286

- Information released by Apple is short and **again** misleading

**Foundation**

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: An application may be able to gain elevated privileges

Description: A memory corruption issue was addressed with improved input validation.


- we only learn that it is some form of memory corruption
- affected component is said to be Foundation
- this hints the vulnerability is located in Foundation.framework (however further analysis will reveal that this is misleading)
Diffing the Framework (I)

- on iOS all built-in frameworks are located in the dyldsharedcache file
- this gigantic file is very hard to work with with most tools
- IDA has meanwhile acceptable support for it
- with a clear diffing target the diff can be performed a lot faster
Diffing the Framework (II)

- unfortunately diffing the Foundation.framework does not reveal any security fixes
- at this point other targets should be diffed
- next obvious choice is CoreFoundation.framework
- the Diaphora result for this diff reveals changes in CFPrefsDaemon
Finding the Vulnerability

• Unfortunately the last CoreFoundation source code on is 4 years old
• that old version does not seem to contain the CFPrefsDaemon code
• furthermore a quick google search did not reveal a copy of the code either
• however ZecOps has performed a manual decompilation of the code in question
Decompiled code (by ZecOps)
Finding the Vulnerability (II)

- In their research ZecOps has identified the vulnerability as a reference counting issue that leads to a double free.
- The issue is in one of the XPC handling functions and therefore triggered via an XPC message.
- The vulnerability is summarised as follows:
  - the code loops through the array `CFPreferencesMessages`.
  - each element is copied into a buffer without increasing references).
  - callback handlers are supposed to retain the object.
  - a crafted XPC message can skip the callback.
  - In this case the object is missing a reference.

```python
poc_dict = {
    "CFPreferencesOperation": 5,
    "CFPreferencesMessages": [
        {
            "CFPreferencesOperation": 4
        }
    ]
}
```
References

Exploitation roadmap for CVE-2019-7287
What kind of vulnerability do we have again?

- we control length of memmove when calling two external methods
- we can copy whatever is behind the input buffer
- overwrite what is behind the output buffer
Where is our input buffer?

- external method is called internally via mach message
- input buffer is inbound in that mach message
- kernel “compresses” message according to parameter length
- position on heap depends on length
Where is our input buffer?

- **ucEncryptSUInfo**
  - 0x7d8 param length
  - message in kalloc.4096 zone

- **ucEncryptWithWrapperKey**
  - 0x8c param length
  - message in kalloc.512 zone

- buffer is in kalloc memory
- attacker can choose what zone is more convenient
Where is our output buffer?

- external method is called internally via mach message
- output buffer is inbound in the reply mach message
- reply message is in kalloc.8192 zone
What kind of vulnerability do we have again?

- source and destination are on different kalloc.X zone
- attacker can freely choose smaller or larger source zone
How to exploit?

- basically just a matter of heap feng shui
- and choosing what data to overwrite with what
- usually target is to somehow end up with a kernel task port

- possible heap feng shui techniques
  - `vm_map_copy_t` [1]
  - `OOL_PORTS_DESCRIPTOR` [2]
  - `OSUnserializeXML / OSUnserializeBinary` [3]
  - `OSUnserializeXML via IOSurface` [4]
  - pre-loaded mach messages for ports [5]
  - ...
How to exploit? (pre iPhone 7 devices)

- on old devices immediate game-over as demonstrated at CanSecWest 2017 [2]
- use `vm_map_copy_t` to put user land pointers behind input buffer
- Use `OOL_PORTS_DESCRIPTOR` to put port pointers behind output buffer
- trigger exploit and use known code
  - CLOCK_PORT
  - TASK_PORT
  - KERNEL_TASK_PORT
How to exploit? (post iPhone 7 devices)

- on new devices exploitation is naturally harder
- however this vulnerability allows a lot of different things

**Example 1:**
- Use `vm_map_copy_t` or `IOSurface` method in output buffer zone
- after memmove content is moved into areas that we can read back
- arbitrary kernel memory info leak (self locate, break KASLR)
How to exploit? (post iPhone 7 devices)

- on new devices exploitation is naturally harder
- however this vulnerability allows a lot of different things

**Example 2:**
- Use `vm_map_copy_t` in input zone to put arbitrary data behind us
- turns our memmove into a “bufferoverflow”
- use exploitation technique from `extract_recipe` for outbuffer side
How to exploit? (post iPhone 7 devices)

- on new devices exploitation is naturally harder
- however this vulnerability allows a lot of different things

- **Example 3:**
  - Use `OOL_PORTS_DESCRIPTOR` in input and output zone
  - copies pointers to legal ports in output buffer
  - after receiving input zone heap feng shui messages all those pointers are dangling
How to exploit? (post iPhone 7 devices)

- on new devices exploitation is naturally harder
- however this vulnerability allows a lot of different things

- **Example 4:**
  - Use `vm_map_copy_t` in input zone to put arbitrary data into behind us
  - turns our memmove into a “bufferoverflow”
  - create many many kernel ports to have them next to output buffer
  - use vulnerability to (partially) overwrite ports
References

- [1] iOS 6 Kernel Security

- [2] Port(al) to the iOS Core
  https://www.slideshare.net/i0n1c/cansecwest-2017-portal-to-the-ios-core

- [3] iOS Kernel Heap Armageddon
  https://media.blackhat.com/bh-us-12/Briefings/Esser/BH_US_12_Esser_iOS_Kernel_Heap_Armageddon_WP.pdf

- [4] Rotten Apples Vulnerability Heaven in the iOS Sandbox

- [5] Ian Beer Slidedeck - link and title missing
Exploitation roadmap for CVE-2019-7286
ZecOps Writeups

- ZecOps also analysed the exploitation of this bug and wrote a write up [1]
- they released POC code
- their POC code does not have actual payload
- the POC does not do exactly what the blog post describes
- it does not actually work on iOS device (didn’t test on Mac)
- wrong addresses and heapspray that kills the device
What kind of vulnerability do we have again?

- reference counting vulnerability that leads to double xpc_release()
- happens in same XPC request without interruption
- no control of memory in the XPC request in between
How to exploit?

• need to race the double free
  • How to fill memory (in between frees)?
  • How to increase race window?

1st xpc_release()

2nd xpc_release()

reallocation
How to fill memory (in between frees)?

- need to create a second thread in daemon
- easiest done by doing another XPC connection
- then need to do XPC heap spraying [2]
  - sending arbitrary XPC arrays

```
1st xpc_release()
2nd xpc_release()
reallocate
```

XPC 1

XPC 2
How to increase race window?

- time between the two frees depends on XPC data
- first free happens in loop over an array `CFPreferencesMessages`
- we can increase race window by adding many values to the array

```c
for( counter = 0; xpc_array_count != counter ; counter++ )
{
    current_element = xpc_buffer[counter];
    if (xpc_get_type(current_element) != &_xpc_type_null )
        xpc_release(current_element);
}
```
What to fill memory with?

- what should we use to replace the freed object with?
- exploitation technique is based on Phrack article by nemo [3]
- need control over first 8 bytes for ISA pointer
- need control over length (0xc0)
- xpc_string is using strdup()
- also can be used many times

- BUT NULL bytes
- POC gives up at this point
How it really worked?

- Google has released on 29th August a description of what really happened [6]
- the exploit is using similar ideas but is different
- not going to copy and paste it here
- just go read blog that seems to be excellent

Explicit flow

The explicit strategy here is to realize the `free'd xpc_dictionary` in the gap between the `xpc_release` when destroying the `sub_message` and the `xpc_release` of the outer request message. They do this by using four threads, running in parallel. Threads A, B and C start up and wait for a global variable to be set to 1. When that happens they each by 100 times to send the following XPC message to the service:

```
[ "CFPreferencesOperation": 5,
  "CFPreferencesMessages" : [10000 * xpc_data_spray ] ]
```

where `xpc_data_spray` is a nil-byte `xpc_data` buffer filled with the operand value 0x13800000. This is the target address to which they will try to heap_spray. They are hoping that the contents of one of these `xpc_data`'s 448-byte backing buffers will overlap with the `free'd xpc_dictionary`, completely filling the memory with the heap_spray address.

As we saw in `CFPreferences.h` `handleMultiMessageReplyHandler` this is not a valid `multiMessage`; the `CFPreferencesMessage` array may only contain dictionaries or NULLs. Nevertheless, it will take some time for all these `xpc_data` objects to be created, `handleMultiMessage` to run, fail and the `xpc_data` objects to be destroyed. They are hoping that with three threads trying this in parallel this replacement strategy will be good enough.

Trigger message

The bug will be triggered by a sub-message with an operation key mapping to a handler which doesn't invoke the reply block. They chose operation `i`, handled by `handleFlushSourceForDomainMessage`. The trigger message looks like this:

```
[ "CFPreferencesOperation": 5
  "CFPreferencesMessages" : [ 8000 * ( op_1_dict, second_op_5_dict ), 150 * (second_op_5_dict, op_4_dict, op_6_dict, op_7_dict), third_op_3_dict ] ]
```

where the sub-message dictionaries are:

```plaintext
op_1_dict = { }
```

© 2019 by ANTIDOTE All rights reserved
Unlimited Tries!

- `cfprefsd` is a LaunchDaemon/Agent
- this means it will be respawned on crash
- while crashing it is noisy we have unlimited tries

- was the original exploit so noisy so that Google noticed?
What todo with ROP?

- `dyld_shared_cache` makes address of all ROP gadgets known to local attackers
- we can create arbitrary ROP programs
- once you can ROP inside `cfprefsd` what can you do?
  - steal its task port to "remote control" it [4] [5]
  - open a driver connection and steal that instead
  - ...

[4] [5]
References

- [1] CVE-2019-7286 Part II: Gaining PC Control

- [2] Auditing and Exploiting Apple IPC
  https://thecyberwire.com/events/docs/IanBeer_JSS_Slides.pdf

  http://phrack.org/issues/69/9.html#article

- [4] An introduction to exploiting userspace race conditions on iOS

- [5] Bypassing platform binary restrictions with task_threads()

- [6] In-the-wild iOS Exploit Chain 4
  https://googleprojectz00.blogspot.com/2019/08/in-wild-ios-exploit-chain-4.html
Conclusion
Conclusion (I)

- both vulnerabilities could be reversed with just a bit of Diaphora
- kernel vulnerability easy to spot from diff
- user space vulnerability took more time to spot because it is more complex
- but this gets easier the more often you do this in code you know
- there are people who do this every day and get paid for just that

- doesn’t really stop attackers for long
Conclusion (II)

- after understanding the vulnerability simple POC exploits can be done fast
- full exploitation takes naturally longer
- the kernel bug felt easier to exploit than the user land bug (more powerful)
- also there is plenty of source code available for iOS kernel exploits
- parts could be cut and pasted
Where is the code?

- The code will show up in the next days on GitHub
  - [https://github.com/AntidOteCom](https://github.com/AntidOteCom)

- Keep updated about the release and other things via Twitter
  - [https://twitter.com/antidOtecom](https://twitter.com/antidOtecom)

- Consider signing up for one of our upcoming trainings
  - [https://www.antidOte.com/stories/training.html](https://www.antidOte.com/stories/training.html)
Questions?

www.antid0te.com

© 2019 by ANTIDOTE. All rights reserved