black hat USA 2020
AUGUST 5-6, 2020
BRIEFINGS

Dive into Apple IO80211FamilyV2

wang yu

About me

wangyu_wy@didiglobal.com

Background of this research project

IO80211Family V1, V2 and Apple 80211 Wi-Fi Subsystem

Introduction

Starting from iOS 13 and macOS 10.15 Catalina, Apple refactored the architecture of the 80211 Wi-Fi client drivers and renamed the new generation design to IO80211FamilyV2.

This presentation will help you better understand the architecture and security challenges of the Apple 80211 Wi-Fi subsystem.

Introduction (cont)

As a Wi-Fi family driver, IO80211Family (V1) / IO80211FamilyV2 plays a key role in Apple's communication model, they manage many important features, such as:

SSID, Channel, Antenna, Rate, TxPower, AP mode and ACL policy settings,
Apple Wireless Direct Link (AWDL) service management,
Background, P2P, Offload scanning,
Troubleshooting, etc.

The era of IO80211Family

IO80211Family (V1) is mainly designed to support Apple Airport and related equipment.

Daemon:	airportd …
Framework:	Apple80211, CoreWifi, CoreWLAN …
_____	
Family drivers:	IO80211Family, IONetworkingFamily
Plugin drivers:	AirPortBrcmNIC, AirPortBrcm4360 / 4331, AirPortAtheros40 …
Low-level drivers:	IOPCIFamily

The era of IO80211FamilyV2

IO80211FamilyV2 is mainly designed for communication and data sharing between new generation mobile-based Apple devices.

Daemon:	airportd …
Framework:	Apple80211, CoreWifi, CoreWLAN …
_____	
Family drivers:	IO80211FamilyV2, IONetworkingFamily
Plugin drivers:	AppleBCMWLANCore replaces AirPort Brcm series drivers
Low-level drivers:	AppleBCMWLANBusInterfacePCIe
Firmware:	BCMWLANFirmware4355 / 4364 / 4377 / 4378 …

The era of IO80211FamilyV2 (cont)

New subsystems are supported, such as Skywalk.

```
11  if ( a3 )
12  {
13    return_value = IO80211Controller::apple80211VirtualRequestIoctl(a1, 0xC03069C9, 0xC, a3, v12);
14  }
15  else if ( a4 )
16  {
17    return_value = (*(*a1 + 0xCC0LL))(a1, 0xC03069C9LL, 0xCLL, a4, v12);// IOSkywalkNetworkInterface
18  }
19  else
20  {
21    return_value = IO80211Controller::apple80211RequestIoctl(a1, 0xC03069C9, 0xC, a2, v12);
22  }
```

IO80211FamilyV2 reverse engineering

New features are supported as well, such as Sidecar.

```
#define APPLE80211_IOC_AWDL_SIDECAR_STATISTICS      0x157
#define APPLE80211_IOC_AWDL_SIDECAR_DIAGNOSTICS     0x15F
```

Summary about the new architecture

IO80211FamilyV2 is a brand new design for the mobile era.

IO80211FamilyV2 and AppleBCMWLANCore integrate the original AirPort Brcm 4331 / 4360 series drivers, with more features and better logic.

Please also keep in mind, new features always mean new attack surfaces.

Where to start?

Can we build a compatible AppleIO80211 or other Wi-Fi framework?

11208elppA
http://newosxbook.com/articles/11208ellpA.html

11208elppA, Part II
http://newosxbook.com/articles/11208ellpA-II.html

I can do this all day

Yeah, I know. But before that, these projects are worth a look:

Intel Wifi for MacOS
https://github.com/AppleIntelWifi

itlwm
https://github.com/OpenIntelWireless/itlwm

Voodoo80211
https://github.com/mercurysquad/Voodoo80211

Attack Surfaces of IO80211Family V1 and V2 Kernel Extensions

Attack surfaces

All inputs are potentially dangerous.

1. From remote and local firmware to operating system kernel

2. From user-mode daemon and framework to operating system kernel

3. All other handlers and parsers for input parameters

From remote and firmware to kernel

AppleBCMWLANCore::handleEventPacket

https://googleprojectzero.blogspot.com/2017/09/over-air-vol-2-pt-1-exploiting-wi-fi.html
https://googleprojectzero.blogspot.com/2017/10/over-air-vol-2-pt-2-exploiting-wi-fi.html
https://googleprojectzero.blogspot.com/2017/10/over-air-vol-2-pt-3-exploiting-wi-fi.html

AppleBCMWLANBusInterfacePCIe::handleFWTrap / CVE-2020-9833

https://support.apple.com/en-us/HT211170

From daemon and framework to kernel

AirPort_Athr5424::setSCAN_REQ
http://www.uninformed.org/?v=all&a=37&t=txt

AirPort_BrcmNIC / IO80211Family Get and Set requests
https://www.zerodayinitiative.com/advisories/ZDI-20-215/
https://www.thezdi.com/blog/2018/10/24/cve-2018-4338-triggering-an-information-disclosure-on-macos-through-a-broadcom-airport-kext

All other handlers and parsers

Protocols such as Apple Wireless Direct Link (AWDL)
https://bugs.chromium.org/p/project-zero/issues/detail?id=1982
https://bugs.chromium.org/p/project-zero/issues/detail?id=2012

Subsystems such as SkyWalk
http://newosxbook.com/bonus/vol1ch16.html

Handlers such as AppleBCMWLANCore::handleDataPacket, etc.

From project Kemon to Wi-Fi subsystem sniffer and fuzzer

Kemon: An Open-Source Pre and Post Callback-Based Framework for macOS Kernel Monitoring
https://github.com/didi/kemon
https://www.blackhat.com/us-18/arsenal/schedule/index.html#kemon-an-open-source-pre-and-post-callback-based-framework-for-macos-kernel-monitoring-12085

The practice of kernel inline hooking
https://www.blackhat.com/us-19/arsenal/schedule/#ksbox-a-fine-grained-macos-malware-sandbox-15059

IO80211Family Get and Set request sniffer

```
[Kemon.kext] : process(pid 198)=mDNSResponder, type=APPLE80211_IOC_AWDL_ELECTION_ALGORITHM_ENABLED, user buffer=0x809b110, length=0x20.
[Kemon.kext] : process(pid 198)=mDNSResponder, type=APPLE80211_IOC_AWDL_ELECTION_ALGORITHM_ENABLED, user buffer=0x809b110, length=0x20.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_RESTORE_DEFAULTS, user buffer=0x1c32b70, length=0x8.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_AWDL_ENABLE_ROAMING, user buffer=0x1d38e88, length=0x930.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_ASSOCIATE, user buffer=0x1c331d8, length=0x1d4.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_AWDL_RSDB_CAPS, user buffer=0x1dbbde0, length=0x4dc.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_AWDL_RSDB_CAPS, user buffer=0x1dbbde0, length=0x4dc.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_RESTORE_DEFAULTS, user buffer=0x1c32c60, length=0x8.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_AWDL_STATISTICS, user buffer=0xe56662b0, length=0x14.
[Kemon.kext] : process(pid 158)=airportd, type=APPLE80211_IOC_SCAN_REQ, user buffer=0x1ec1678, length=0x954.
[Kemon.kext] : process(pid 198)=mDNSResponder, type=APPLE80211_IOC_AWDL_ELECTION_ALGORITHM_ENABLED, user buffer=0x809beb0, length=0x20.
[Kemon.kext] : process(pid 198)=mDNSResponder, type=APPLE80211_IOC_AWDL_ELECTION_ALGORITHM_ENABLED, user buffer=0x809bfc0, length=0x20.
[Kemon.kext] : process(pid 198)=mDNSResponder, type=APPLE80211_IOC_AWDL_ELECTION_ALGORITHM_ENABLED, user buffer=0x809afc0, length=0x20.
```

Kemon-based sniffer

Apple 80211 Wi-Fi subsystem fuzzer

Code coverage analysis based on Kemon's kernel inline hook engine.

Passive fuzzing based on Wi-Fi sniffer and active fuzzing based on compatible framework.

Combining the two fuzzing methods.

IO80211Family V1 and V2
Latest Zero-day Vulnerability Case Studies

Binary auditing and vulnerability hunting

The total number of reported vulnerabilities:
Eighteen. Four of them were patched on May 26, 2020. (before WWDC20)

The types of vulnerabilities include:
1. Heap overflow / kernel object out-of-bounds write
2. Heap data out-of-bounds access
3. Kernel information disclosure
4. Stack overflow without canary protection
5. Arbitrary kernel memory write
6. Integer overflow / unsigned vs signed comparison, etc.

Vulnerability classification

Zero-days can be classified into at least three categories from the high level of the architecture:

1. Vulnerabilities affecting only IO80211FamilyV2

2. Vulnerabilities affecting both IO80211Family (V1) and IO80211FamilyV2

3. Vulnerabilities affecting only IO80211Family (V1)

Vulnerability classification (cont)

Zero-days can be classified into at least three categories from the high level of the architecture:

1. Vulnerabilities affecting only IO80211FamilyV2
   1.1. Introduced when porting existing V1 features
   1.2. Introduced when implementing new V2 features

2. Vulnerabilities affecting both IO80211Family (V1) and IO80211FamilyV2

3. Vulnerabilities affecting only IO80211Family (V1)

Category 1.1 – Introduced into V2 when porting existing V1 features

CVE-2020-9834:
AppleBCMWLANCore`AppleBCMWLANScanManager::fillScanParams
Kernel Object Out-of-bounds Write Vulnerability

Patched via Security Update 2020-003
https://support.apple.com/en-us/HT211170

Random panic case one – Hah?

```
(lldb) di -p
kernel`_delayed_call_enqueue:
->  0xffffff8000763a2a <+538>: cmpq   %rax, 0x8(%rbx)
    0xffffff8000763a2e <+542>: jne    0xffffff8000763adb
    0xffffff8000763a34 <+548>: cmpq   %rax, (%r11)
    0xffffff8000763a37 <+551>: jne    0xffffff8000763adb

(lldb) register read rbx
     rbx = 0x0000000000000000

(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff8000763a2a kernel`_delayed_call_enqueue [inlined] at queue.h:245 [opt]
    frame #1: 0xffffff8000763a02 kernel`_delayed_call_enqueue [inlined] at queue.h:351 [opt]
    frame #2: 0xffffff8000763a02 kernel`_delayed_call_enqueue [inlined] at call_entry.h:150 [opt]
    frame #3: 0xffffff80007639a5 kernel`_delayed_call_enqueue at thread_call.c:523 [opt]
    frame #4: 0xffffff80007640d8 kernel`thread_call_enter_delayed_internal at thread_call.c:1079 [opt]
    frame #5: 0xffffff80007384e2 kernel`mk_timer_arm_trap_internal [inlined] at thread_call.c:994 [opt]
    frame #6: ......
```

Random panic case two – NULL pointer dereference (again)?

```
(lldb) di -p
AppleBCMWLANCore`AppleBCMWLANHistogram::dump:
->  0xffffff7f86aee881 <+129>: movl   (%rax,%rcx,4), %ecx
    0xffffff7f86aee884 <+132>: leaq   0x7bb69(%rip), %rdx       ; "%u,"
    0xffffff7f86aee88b <+139>: xorl   %eax, %eax

(lldb) register read rax rcx
       rax = 0x0000000000000000
       rcx = 0x0000000000000000

(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff7f86aee881 AppleBCMWLANCore`AppleBCMWLANHistogram::dump + 129
    frame #1: 0xffffff7f86a6e60f AppleBCMWLANCore`AppleBCMWLANCore::printDataPathDebug + 1939
    frame #2: 0xffffff7f86aa524c AppleBCMWLANCore`AppleBCMWLANCore::captureDriverState + 1564
    frame #3: 0xffffff7f86a53fd6 AppleBCMWLANCore`AppleBCMWLANCore::collectImmediateFaultDataCallback + 108
    frame #4: 0xffffff7f85b36f44 corecapture`CCFaultReporter::collectImmediateData + 72
    frame #5: 0xffffff7f85b37391 corecapture`CCFaultReporter::processFault + 339
    frame #6: ......
```

Random panic case three – Exploitable?

```
(lldb) di -p
kernel`build_path_with_parent:
->  0xffffff80099637e0 <+592>: movb   (%rdx), %al
    0xffffff80099637e2 <+594>: movq   %rdx, %rsi
    0xffffff80099637e5 <+597>: testb  %al, %al

(lldb) register read rdx
       rdx = 0x0000656b00000000

(lldb) bt
* thread #1, stop reason = EXC_BAD_ACCESS (code=1, address=0x0)
  * frame #0: 0xffffff80099637e0 kernel`build_path_with_parent at vfs_cache.c:542 [opt]
    frame #1: 0xffffff8009c198b9 kernel`audit_canon_path [inlined] at vfs_cache.c:801 [opt]
    frame #2: 0xffffff8009c19896 kernel`audit_canon_path [inlined] at vfs_subr.c:2851 [opt]
    frame #3: 0xffffff8009c19891 kernel`audit_canon_path at audit_bsm_klib.c:853 [opt]
    frame #4: 0xffffff8009c0ef5b kernel`audit_arg_sockaddr [inlined] at audit_arg.c:671 [opt]
    frame #5: 0xffffff8009c0eeea kernel`audit_arg_sockaddr at audit_arg.c:373 [opt]
    frame #6: ......
```

Random panic case four – Hmmm, looks like exploitable

```
(lldb) di -p
kernel`OSMetaClassBase::safeMetaCast:
->  0xffffff800c595355 <+21>: callq  *0x38(%rax)
    0xffffff800c595358 <+24>: nopl   (%rax,%rax)
    0xffffff800c595360 <+32>: cmpq   %rbx, %rax

(lldb) register read rax
       rax = 0x6742040014004232

(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff800c595355 kernel`OSMetaClassBase::safeMetaCast [inlined] at OSMetaClass.cpp:1362 [opt]
    frame #1: 0xffffff800c595352 kernel`OSMetaClassBase::safeMetaCast [inlined] at OSMetaClass.cpp:375 [opt]
    frame #2: 0xffffff800c595352 kernel`OSMetaClassBase::safeMetaCast at OSMetaClass.cpp:283 [opt]
    frame #3: 0xffffff7f8e2a4da9 AppleBCMWLANCore`AppleBCMWLANCore::captureDriverState + 377
    frame #4: 0xffffff7f8e253fd6 AppleBCMWLANCore`AppleBCMWLANCore::collectImmediateFaultDataCallback + 108
    frame #5: 0xffffff7f8d336f44 corecapture`CCFaultReporter::collectImmediateData + 72
    frame #6: ......
```

Routine setScanRequest

From IO80211FamilyV2`setScanRequest
to AppleBCMWLANCore`AppleBCMWLANCore::setSCAN_REQ
and then, to AppleBCMWLANCore`AppleBCMWLANScanManager::fillScanParams.

Reverse engineering shows that
the input structure should not be
greater than 0x9D4.

So, can we get the parameter details of the
input structure through reverse engineering?

```
23      req_length = req_set->req_length;
24      return_value = 0x16;
25
26      if ( req_length )
27      {
28        req_data = req_set->req_data;
29        if ( req_data )
30        {
31          //
32          // The length of the input structure is limited to 0x9D4
33          //
34          tmp_length = 0x9D4LL;
35          if ( req_length < 0x9D4 )
36            tmp_length = req_length;
37
38          return_value = IO80211Controller::copyIn(this, req_data, v9, tmp_length);
```

IO80211FamilyV2`setScanRequest reverse engineering

Reverse engineering

With the help of CCLogStream debugging information we can at least identify the offsets at 0x04, 0x10, 0x14, 0x34 and 0x44.

	0	4	8	C	F
0x00	version	bss_type			
0x10	ssid_len	ssid 1	2	3	
0x20	4	5	6	7	
0x30	8	scan_type			
0x40		num_channels	channel data 1	2	
0x50	3				
0x60					
0x70					

Captured from user input data structure

```
CCLogStream::logNoticeIf(
  v17,
  0x8040uLL,
  "%s@%d: [%s]: scan_type = %d, bss_type = %d, num_channels = %u, ssid = \"%s\"\n",
  "setSCAN_REQ",
  21586LL,
  proc_selfname,
  *(input_structure + 0x34),
  *(input_structure + 4),
  *(input_structure + 0x44),
  input_structure_offset_0x14);
```

```
::proc_selfname(proc_selfname, 16);
if ( *(input_structure + 0x10) < 0x21u )
{
  ......
}
else
{
  v5 = *(this + 1290);
  return_value = 0xE0000001;
  if ( v5 && *(*(v5 + 39) + 8LL) >= 2 )
    CCLogStream::logCrit(v5, "%s@%d: [%s]: Unexpected ssid len(%d, %d) \n", "setSCAN_REQ", 21545LL, proc_selfname);
}
```

IO80211FamilyV2`setScanRequest reverse engineering

Routine AppleBCMWLANScanManager::fillScanParams

	0	4	8	C	F
0x00	version	bss_type	ether_addr / ether_null		pad
0x10	ssid_len	ssid 1	2		3
0x20	4	5	6		7
0x30	8	scan_type			
0x40		num_channels	channel data 1		2
0x50	3				
0x60					
0x70					
0x80					
0x90					
0xA0					

Captured from user input data structure

filling

at object offset 0x5CC offset 0x5D4

	0	4	8	C	F
0x00	0x00000001	0x0001		ssid_len	ssid 1
0x10	2	3		4	5
0x20	6	7		8	ether_addr
0x30	ether_addr / bss type / scan type	0xFFFFFFFF		0xFFFFFFFF / user input+0x3C	0xFFFFFFFF / user input+0x3C
0x40	0xFFFFFFFF / user input+0x40	num_channels		channel data	
0x50					
0x60					

AppleBCMWLANScanManager's internal object

Routine AppleBCMWLANScanManager::fillScanParams (cont)

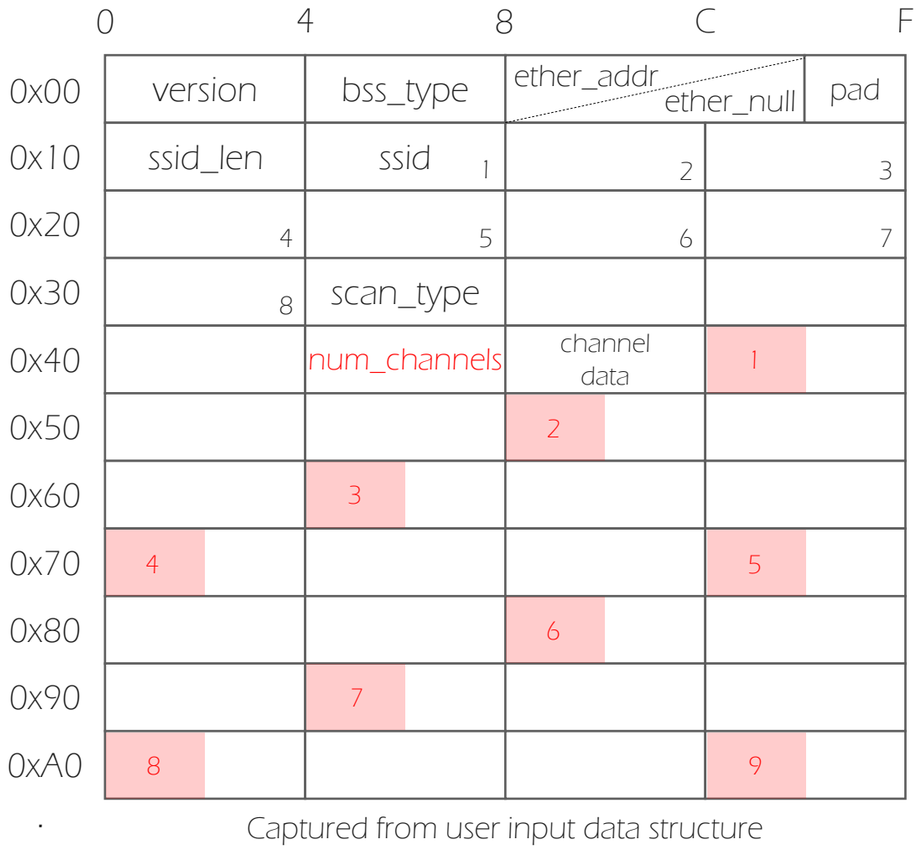
	0	4	8	C	F
0x00	version	bss_type	ether_addr / ether_null		pad
0x10	ssid_len	ssid 1	2		3
0x20	4	5	6		7
0x30	8	scan_type			
0x40		num_channels	channel data 1		2
0x50	3				
0x60					
0x70					
0x80					
0x90					
0xA0					

Captured from user input data structure

```
179  num_channels = *(input_structure + 0x44);
180  if ( *(input_structure + 0x44) )
181  {
182    *(internal_object + 0xF) = num_channels;     // at offset 0x3C
183    source = (input_structure + 0x4C);
184    index = 0LL;
185    do
186    {
187      *&internal_object[2 * index++ + 0x40] = *source;
188      source += 0xC;
189    }
190    while ( num_channels != index );             // no limit?
191  }
```

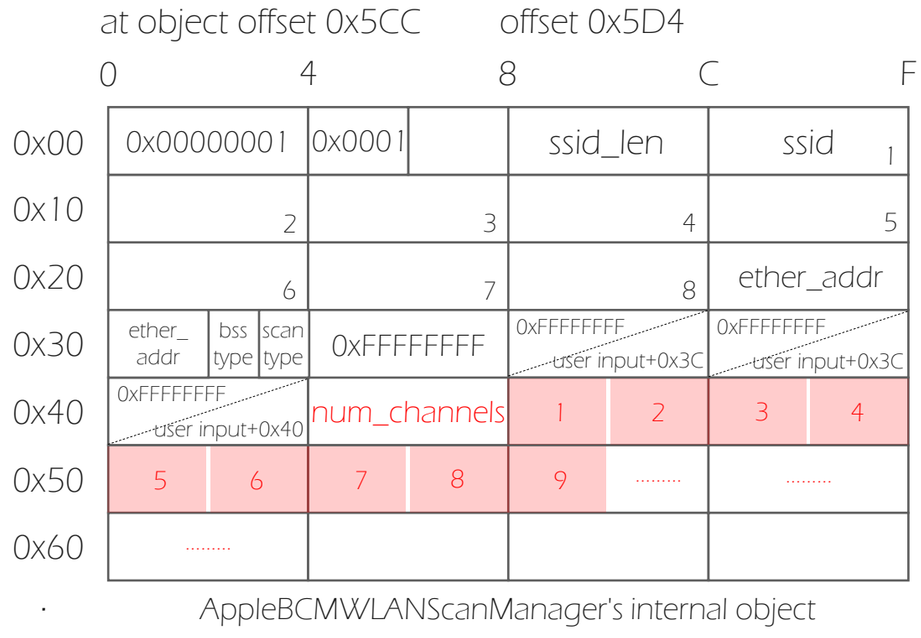
AppleBCMWLANScanManager::fillScanParams reverse engineering

AppleBCMWLANScanManager::fillScanParams heap overflow



Captured from user input data structure

filling



AppleBCMWLANScanManager's internal object

The root cause of CVE-2020-9834

The vulnerable function lacks the necessary checks for num_channels in the input structure, which leads to out-of-bounds operations.

For source buffer this means out-of-bounds access, and for destination buffer this means heap overwrite.

The good news is that the write primitive is relatively complete.
For vulnerability with incomplete write primitives, please refer to:
https://www.blackhat.com/docs/asia-16/materials/asia-16-Wang-A-New-CVE-2015-0057-Exploit-Technology-wp.pdf

There are still many questions

We have identified the root cause of the vulnerability, but:

1. What is the meaning of the remaining fields in the input structure?

2. Why does the write primitive read from the inputs every 0x0c bytes?

Apple SDKs

Apple80211 SDKs (for 10.4 Tiger, 10.5 Leopard and 10.6 Snow Leopard)
https://github.com/phracker/MacOSX-SDKs/releases

IO80211Interface / IO80211Controller finally (somewhat?) open
https://lists.apple.com/archives/xcode-users/2007/Nov/msg00544.html

```
apple80211_ioctl.h
apple80211_var.h
apple80211_wps.h
IO80211Controller.h
IO80211Interface.h
IO80211WorkLoop.h
```

Apple SDKs leaked? Really?

Apple80211 SDK of macOS 10.12 Sierra
https://github.com/rpeshkov/black80211/tree/master/Black80211/apple80211/sierra

Apple80211 SDK of macOS 10.13 High Sierra
https://github.com/rpeshkov/black80211/tree/master/Black80211/apple80211/high_sierra

Apple80211 SDK of macOS 10.15 Catalina
https://github.com/AppleIntelWifi/Black80211-Catalina/tree/master/Black80211/apple80211/catalina

New features and interfaces based on reverse engineering

```
#define APPLE80211_IOC_AWDL_PEERS_INFO           0xFA
#define APPLE80211_IOC_TKO_PARAMS                0xFB
#define APPLE80211_IOC_TKO_DUMP                  0xFC
#define APPLE80211_IOC_AWDL_NEARBY_LOG_TRIGGER   0xFD
#define APPLE80211_IOC_HW_SUPPORTED_CHANNELS     0xFE
#define APPLE80211_IOC_BTCOEX_PROFILE            0xFF
#define APPLE80211_IOC_BTCOEX_PROFILE_ACTIVE     0x100
#define APPLE80211_IOC_TRAP_INFO                 0x101
#define APPLE80211_IOC_THERMAL_INDEX             0x102
#define APPLE80211_IOC_MAX_NSS_FOR_AP            0x103
#define APPLE80211_IOC_BTCOEX_2G_CHAIN_DISABLE   0x104
#define APPLE80211_IOC_POWER_BUDGET              0x105
#define APPLE80211_IOC_AWDL_DFSP_CONFIG          0x106
#define APPLE80211_IOC_AWDL_DFSP_UCSA_CONFIG     0x107
#define APPLE80211_IOC_SCAN_BACKOFF_REPORT       0x108
#define APPLE80211_IOC_OFFLOAD_TCPKA_ENABLE      0x109
#define APPLE80211_IOC_RANGING_CAPS              0x10A
#define APPLE80211_IOC_PER_CORE_RSSI_REPORT      0x10B
```

Giving back to the community

```
#define APPLE80211_IOC_COMPANION_SKYWALK_LINK_STATE             0x162
#define APPLE80211_IOC_NAN_LLW_PARAMS                           0x163
#define APPLE80211_IOC_HP2P_CAPS                                0x164
#define APPLE80211_IOC_RLLW_STATS                               0x165
        APPLE80211_IOC_UNKNOWN (NULL/No corresponding handler)  0x166
#define APPLE80211_IOC_HW_ADDR                                  0x167
#define APPLE80211_IOC_SCAN_CONTROL                             0x168
        APPLE80211_IOC_UNKNOWN (NULL/No corresponding handler)  0x169
#define APPLE80211_IOC_CHIP_DIAGS                               0x16A
#define APPLE80211_IOC_USB_HOST_NOTIFICATION                    0x16B
#define APPLE80211_IOC_LOWLATENCY_STATISTICS                    0x16C
#define APPLE80211_IOC_DISPLAY_STATE                            0x16D
#define APPLE80211_IOC_NAN_OOB_AF_TX                            0x16E
#define APPLE80211_IOC_NAN_DATA_PATH_KEEP_ALIVE_IDENTIFIER      0x16F
#define APPLE80211_IOC_SET_MAC_ADDRESS                          0x170
#define APPLE80211_IOC_ASSOCIATE_EXTENDED_RESULT                0x171
#define APPLE80211_IOC_AWDL_AIRPLAY_STATISTICS                  0x172
#define APPLE80211_IOC_HP2P_CTRL                                0x173
#define APPLE80211_IOC_REQUEST_BSS_BLACKLIST                    0x174
#define APPLE80211_IOC_ASSOC_READY_STATUS                       0x175
#define APPLE80211_IOC_TXRX_CHAIN_INFO                          0x176
```

This is the story behind

	0	4	8	C	F
0x00	version	bss_type	bssid		pad
0x10	ssid_len	ssid 1	2	3	
0x20	4	5	6	7	
0x30	8	scan_type	phy_mode	dwell_time	
0x40	rest_time	num_channels	version	channel	
0x50	flags	version	channel	flags	
0x60	version	channel	flags	version	
0x70	channel	flags	version	channel	
0x80	flags	version	channel	flags	
0x90	version	channel	flags	version	
0xA0	channel	flags	version	channel	

Captured from user input data structure

```
#define APPLE80211_VERSION 1

#define APPLE80211_MAX_SSID_LEN 32
#define APPLE80211_MAX_CHANNELS 64 // Please note that the array size
                                   // should be limited to MAX_CHANNELS
struct apple80211_channel
{
    u_int32_t version;
    u_int32_t channel;              // channel number
    u_int32_t flags;                // apple80211_channel_flag vector
};

struct apple80211_scan_data
{
    u_int32_t version;
    u_int32_t bss_type;             // apple80211_apmode
    struct ether_addr bssid;        // target BSSID
    u_int32_t ssid_len;             // length of the SSID
    u_int8_t ssid[APPLE80211_MAX_SSID_LEN];
    u_int32_t scan_type;            // apple80211_scan_type
    u_int32_t phy_mode;             // apple80211_phymode vector
    u_int16_t dwell_time;           // time to spend on each channel (ms)
    u_int32_t rest_time;            // time between scanning each channel (ms)
    u_int32_t num_channels;         // 0 if not passing in channels
    struct apple80211_channel channels[APPLE80211_MAX_CHANNELS]; // channel list
};
```

apple80211_ioctl.h and apple80211_var.h

One more question

Why can such an obvious vulnerability survive to 2020?

The answer

The answer is that this vulnerability was introduced into IO80211FamilyV2 (iOS 13 and macOS 10.15 Catalina) when porting the existing features of V1, and there is no problem with V1 related function.

For unknown reasons, IO80211FamilyV2 removed this boundary check.

```
channels = (input_structure_offset_0x48 + 4);
bss_type = 0LL;
index = 0LL;
do
{
  flag = 0xC000;
  if ( *channels < 0xFu )
    flag = 0;
  *(internal_object + 2 * index++) = *channels | flag | 0x1000;
  if ( index >= number_of_channels )
    break;
  channels += 0xC;
}
while ( index < 0x80 );                 // The array size is limited to 0x80
```

AirPortBrcmNIC`AirPort_BrcmNIC::scanreq_common reverse engineering

Back to the future

OS X Kernel-mode Exploitation in a Weekend, September 2007.
http://www.uninformed.org/?v=all&a=37&t=txt

From AirPort_Athr5424::setSCAN_REQ to AppleBCMWLANCore::setSCAN_REQ:

```
Program received signal SIGTRAP, Trace/breakpoint trap.
0x001933de in memcpy_common ()
2: x/i $eip  0x1933de <memcpy_common+10>:    repz movs DWORD PTR es:[edi],DWORD PTR ds:[esi]
#0  0x001933de in memcpy_common ()
#1  0x03915004 in ?? ()
#2  0x008c6083 in sta_iterate ()
#3  0x008e52b7 in AirPort_Athr5424::ieee80211_notify_scan_done ()
#4  0x008e55b9 in AirPort_Athr5424::setSCAN_REQ ()
#5  0x008b2c91 in IO80211Scanner::scan ()
#6  0x008aa00c in IO80211Controller::execCommand ()
#7  0x0038e698 in IOCommandGate::runAction ()
```

Category 1.2 – Introduced into V2 when implementing new features

CVE-2020-9833:
AppleBCMWLANBusInterfacePCIe::loadChipImage /
AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob
Kernel Information Disclosure Vulnerability

Patched via Security Update 2020-003
https://support.apple.com/en-us/HT211170

Reverse engineering and binary auditing

Step 1. Allocation but not initialized

```
if ( !*(this + 0x230) )
{
  trap_info_buffer = IOMalloc(0x208uLL);
  v65 = this;
  *(this + 0x230) = trap_info_buffer;
  if ( !trap_info_buffer )
  {
    v140 = *v269;
    v9 = 0xE00002BD;
    if ( *v269 && *(*(v140 + 39) + 8LL) > 0 )
      CCLogStream::logAlert(
        v140,
        "%s@%d:Failed to allocate trap info buffer\n",
        "createFirmwarePCIeIPC",
        6566LL);
    goto LABEL_137;
  }
  v79 = *(this + 146);
}
```

AppleBCMWLANBusInterfacePCIe::loadChipImage reverse engineering

Step 2. Initialization

```
if ( *(this + 0x1175) )
{
  trap_info_buffer = *(this + 0x230);
  firmware_trap = ((*(**(*(this + 0x6A) + 32LL) + 744LL))(*(*(this + 0x6A) + 32LL), 0LL, *(*(this + 0x6A) + 64LL)) + 4);
  trap_info_length = 0x204LL;
  if ( !*(this + 0x1175) )
    trap_info_length = 0LL;
  memcpy(trap_info_buffer, firmware_trap, trap_info_length);
}
```

AppleBCMWLANBusInterfacePCIe::handleFWTrap reverse engineering

Step 3. Firmware trap info extraction

```
trap_info_length = 0x204LL;
if ( !*(this + 0x1175) )
  trap_info_length = 0LL;
result = 0xE00002BCLL;
if ( trap_info_length >= output_length )
{
  memcpy(output_buffer, *(this + 0x230), output_length);
  result = 0LL;
}
```

AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob reverse engineering

Bypass the AppleBCMWLANBusInterfacePCIe::handleFWTrap

The expected execution order is Step 1, 2 and then 3.

Is it possible to extract information in the trap buffer before it is initialized?
Is it possible to "race" the execution order from Step 1, 2 and 3 to Step 1, 3, (2)?

Yes, It is possible

The leaked heap data can exceed 0x200 bytes.

Including, kernel objects, function pointers, etc.

```
didi@didiMacBook-Pro Desktop % ./leak
                                -*> MEMORY DUMP <*-
+--------------------+-------------------------------------------------+------------------+
|      ADDRESS       |  0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F | 0123456789ABCDEF |
| ------------------ +-------------------------------------------------+ ---------------- |
|  0x00007ffeefbf8860 | 01 00 00 00 00 00 00 00  00 00 00 00 26 bc 33 ae | ............&.3. |
|  0x00007ffeefbf8870 | 20 4e 0a 73 40 72 17 46  80 ff ff ff 40 72 17 46 |  N.s@r.F....@r.F |
|  0x00007ffeefbf8880 | 80 ff ff ff f4 72 17 46  80 ff ff ff 00 00 00 00 | .....r.F........ |
|  0x00007ffeefbf8890 | 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffeefbf88a0 | 00 00 00 00 00 00 00 00  00 00 00 00 65 00 78 00 | ............e.x. |
|  0x00007ffeefbf88b0 | 74 00 65 00 6e 00 73 00  69 00 6f 00 6e 00 73 00 | t.e.n.s.i.o.n.s. |
|  0x00007ffeefbf88c0 | 2f 00 69 00 6f 00 75 00  73 00 62 00 68 00 6f 00 | /.i.o.u.s.b.h.o. |
|  0x00007ffeefbf88d0 | 73 00 74 00 66 00 61 00  6d 00 69 00 6c 00 79 00 | s.t.f.a.m.i.l.y. |
|  0x00007ffeefbf88e0 | 2e 00 6b 00 65 00 78 00  74 00 2f 00 63 00 6f 00 | ..k.e.x.t./.c.o. |
|  0x00007ffeefbf88f0 | 6e 00 74 00 65 00 6e 00  74 00 73 00 b8 c9 9c 97 | n.t.e.n.t.s..... |
|  0x00007ffeefbf8900 | 7f ff ff ff e0 c9 9c 97  7f ff ff ff 08 ca 9c 97 | ................ |
|  0x00007ffeefbf8910 | 7f ff ff ff 12 11 11 80  12 11 11 80 24 01 00 00 | ............$... |
|  0x00007ffeefbf8920 | 12 11 11 80 1c 01 00 00  43 fe 02 00 03 13 00 00 | ........C....... |
|  0x00007ffeefbf8930 | 3b 00 03 00 00 00 00 10  01 00 00 00 03 13 00 00 | ;............... |
|  0x00007ffeefbf8940 | 00 00 00 00 00 00 11 00  43 50 58 40 05 00 00 00 | ........CPX@.... |
|  0x00007ffeefbf8950 | 00 f0 00 00 e4 00 00 00  09 00 00 00 73 75 62 73 | ............subs |
|  0x00007ffeefbf8960 | 79 73 74 65 6d 00 00 00  00 40 00 00 05 00 00 00 | ystem....@...... |
|  0x00007ffeefbf8970 | 00 00 00 00 68 61 6e 64  6c 65 00 00 00 40 00 00 | ....handle...@.. |
|  0x00007ffeefbf8980 | 00 00 00 00 00 00 00 00  69 6e 73 74 61 6e 63 65 | ........instance |
|  0x00007ffeefbf8990 | 00 00 00 00 00 a0 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffeefbf89a0 | 00 00 00 00 00 00 00 00  72 6f 75 74 69 6e 65 00 | ........routine. |
|  0x00007ffeefbf89b0 | 00 40 00 00 cf 00 00 00  00 00 00 00 66 6c 61 67 | .@..........flag |
|  0x00007ffeefbf89c0 | 73 00 00 00 00 40 00 00  00 00 00 00 00 00 00 00 | s....@.......... |
|  0x00007ffeefbf89d0 | 6e 61 6d 65 00 00 00 00  00 90 00 00 1e 00 00 00 | name............ |
|  0x00007ffeefbf89e0 | 63 6f 6d 2e 61 70 70 6c  65 2e 77 69 6e 64 6f 77 | com.apple.window |
|  0x00007ffeefbf89f0 | 73 65 72 76 65 72 2e 61  63 74 69 76 65 00 00 00 | server.active... |
|  0x00007ffeefbf8a00 | 74 79 70 65 00 00 00 00  00 40 00 00 07 00 00 00 | type.....@...... |
|  0x00007ffeefbf8a10 | 00 00 00 00 74 61 72 67  65 74 70 69 64 00 00 00 | ....targetpid... |
|  0x00007ffeefbf8a20 | 00 30 00 00 00 00 00 00  00 00 00 00 64 6f 6d 61 | .0..........doma |
|  0x00007ffeefbf8a30 | 69 6e 2d 70 6f 72 74 00  00 d0 00 00 00 00 00 00 | in-port......... |
|  0x00007ffeefbf8a40 | 3c 00 00 00 80 05 00 00  00 00 00 00 00 00 00 00 | <............... |
|  0x00007ffeefbf8a50 | ff ff ff ff 00 00 00 00  00 00 00 00 00 00 00 00 | ................ |
|  0x00007ffeefbf8a60 | 00 00 00 00                                     | ....             |
+--------------------+-------------------------------------------------+------------------+
```

Defeat KASLR

```
Process 1 stopped
* thread #1, stop reason = instruction step over
    frame #0: 0xffffff7f8503897c AppleBCMWLANBusInterfacePCIe`AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob(unsigned char*, unsigned long) + 80
AppleBCMWLANBusInterfacePCIe`AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob:
->  0xffffff7f8503897c <+80>: callq  0xffffff8002998050        ; memcpy
    0xffffff7f85038981 <+85>: xorl   %eax, %eax
    0xffffff7f85038983 <+87>: addq   $0x8, %rsp
    0xffffff7f85038987 <+91>: popq   %rbx
(lldb) register read rdi rsi rdx
       rdi = 0xffffff8200a7376c      // destination
       rsi = 0xffffff8035191000      // source
       rdx = 0x0000000000000204      // num
(lldb) n
Process 1 stopped
* thread #1, stop reason = instruction step over
    frame #0: 0xffffff7f85038981 AppleBCMWLANBusInterfacePCIe`AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob(unsigned char*, unsigned long) + 85
AppleBCMWLANBusInterfacePCIe`AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob:
->  0xffffff7f85038981 <+85>: xorl   %eax, %eax
    0xffffff7f85038983 <+87>: addq   $0x8, %rsp
    0xffffff7f85038987 <+91>: popq   %rbx
    0xffffff7f85038988 <+92>: popq   %r14
(lldb) memory read 0xffffff8200a7376c -c0x204
0xffffff8200a7376c: 23 fa e8 2c ac f9 19 47 00 10 19 35 80 ff ff ff  #??,??.G...5.???
0xffffff8200a7377c: 00 10 19 35 80 ff ff ff b4 10 19 35 80 ff ff ff  ...5.????..5.???
0xffffff8200a7378c: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xffffff8200a7379c: ff 10 00 00 ff 10 00 00 00 00 00 00 00 00 00 00  ?...?...........
0xffffff8200a737ac: 69 00 6e 00 70 00 75 00 74 00 20 00 6d 00 65 00  i.n.p.u.t. .m.e.
0xffffff8200a737bc: 74 00 68 00 6f 00 64 00 73 00 2f 00 74 00 63 00  t.h.o.d.s./.t.c.
0xffffff8200a737cc: 69 00 6d 00 2e 00 61 00 70 00 70 00 2f 00 63 00  i.m...a.p.p./.c.
0xffffff8200a737dc: 6f 00 6e 00 74 00 65 00 6e 00 74 00 73 00 2f 00  o.n.t.e.n.t.s./.
```

"trap info" heap base address is leaked

DEMO

CVE-2020-9833:
AppleBCMWLANBusInterfacePCIe::loadChipImage /
AppleBCMWLANBusInterfacePCIe::copyTrapInfoBlob
Kernel Information Disclosure Vulnerability

Category 2 – Affecting both IO80211Family (V1) and IO80211FamilyV2

Follow-up ID 729483413 / CVE-2020-9832:
IO80211Family`IO80211PeerManager::setScanningState
OOB Access Vulnerability

Follow-up ID 729476502:
IO80211FamilyV2`IO80211PeerManager::setScanningState
OOB Access Vulnerability

Patched via Security Update 2020-003
https://support.apple.com/en-us/HT211170

Routine IO80211PeerManager::setScanningState

```
(lldb) di -p
IO80211Family`IO80211PeerManager::setScanningState:
->  0xffffff7f89492902 <+60>: cmpl   $0x0, (%rdi)
(lldb) register read rax rbx rdi
     rax = 0x00000000de22b04b
     rbx = 0x00000000deadbeef
     rdi = 0xffffff8034975000
(lldb) bt
* thread #1, stop reason = EXC_BAD_ACCESS (code=1, address=0x34975000)
  * frame #0: 0xffffff7f89492902 IO80211Family`IO80211PeerManager::setScanningState + 60

(lldb) di -p
IO80211FamilyV2`IO80211PeerManager::setScanningState:
->  0xffffff7f87212c83 <+57>: cmpl   $0xb, 0x4c(%rcx,%rbx)
(lldb) register read rax rbx rcx
     rax = 0x00000000deadbeef
     rbx = 0x0000000018dbcfb4
     rcx = 0xffffff803b993000
(lldb) bt
* thread #1, stop reason = EXC_BAD_ACCESS (code=1, address=0x54750000)
  * frame #0: 0xffffff7f87212c83 IO80211FamilyV2`IO80211PeerManager::setScanningState + 57
```

The root cause of CVE-2020-9832

Both IO80211Family (V1) and IO80211FamilyV2 made mistakes when checking input parameters.

This vulnerability can be used to detect and analyze kernel heap data or layout, but its quality cannot be compared with CVE-2020-9833.

Category 3 – Vulnerabilities affecting only V1

IO80211FamilyV2 fixes vulnerable functions.
Unfortunately, these important improvements have not been synchronized with other system platforms, so we can use them to attack targets like the latest macOS Mojave (10.14.6 18G5033) and macOS High Sierra (10.13.5 17G13035).

Follow-up ID 729885295:
Apple plans to address this vulnerability in a future security update.

More vulnerabilities

There are still many interesting and powerful vulnerabilities waiting to be fixed.
In the future, I will share their technical details via blog.

Let's protect the endpoint security of Apple platforms together!

DEMO

Apple 80211 Wi-Fi Subsystem Fuzzer on macOS 11.0 Big Sur

Entitlement for iOS

"... as a result of the ioctl(2) failing (with errno/perror(2) reporting - ENOTSUPP/"Operation not supported on socket").
This can be fixed by granting us the same entitlements that /usr/sbin/wifid itself possesses. Specifically, the following:"

Apple 80211 - 28 Days Later (a.k.a 11201ellpA, Part II)

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.wlan.authentication</key>
        <true/>
</dict>
</plist>
```

Trigger CVE-2020-9834 on iOS platform

```
{"bug_type":"210","timestamp":"2020-06-07 15:13:44.99 +0800","os_version":"iPhone OS 13.3.1 (17D50)",}
{
    "build"   : "iPhone OS 13.3.1 (17D50)",
    "product" : "iPhone12,3",
    "kernel"  : "Darwin Kernel Version 19.3.0:
                 Thu Jan  9 21:11:10 PST 2020; root:xnu-6153.82.3~1\/RELEASE_ARM64_T8030",
    "date"    : "2020-06-07 15:13:42.61 +0800",

    "panicString" : "panic(cpu 1 caller 0xffffffff0194c76dc):
    Kernel data abort. at pc 0xffffffff01942b96c, lr 0x72f800701a235570 (saved state: 0xffffffe066d4a150)
    x0: 0xffffffe000c2d000  x1:  0x0000000000000001  x2:  0x0000000000000001  x3:  0x0000000000000001
    x4: 0xffffffe066d4a8a0  x5:  0xffffffe066d4a758  x6:  0x0000000000001000  x7:  0xffffffe0073c3780
    x8: 0xffffffe0014e8600  x9:  0x989b636263620038  x10: 0x0000000000000038  x11: 0x0000000000000100
    x12: 0x0000000000004000 x13: 0x0000000000000000  x14: 0x0000000000000000  x15: 0x0000000000227277
    x16: 0xffffffff018e39e80 x17: 0xffffffff018e39e80  x18: 0x0000000000000000  x19: 0xffffffe000c2d000
    x20: 0x0000000000000001 x21: 0xffffffe0073c3780  x22: 0x0000000003a7d656  x23: 0xffffffe066d4a758
    x24: 0x0000000000000007 x25: 0x0000000000000001  x26: 0x0000000000000002  x27: 0xffffffe066d4a8a0
    x28: 0xffffffe00137b190 fp:  0xffffffe066d4a4c0  lr:  0x72f800701a235570  sp:  0xffffffe066d4a4a0
    pc:  0xffffffff01942b96c cpsr: 0x20400304        esr: 0x96000004          far: 0x989b636263620060
```

Takeaways and The End

Takeaways

IO80211FamilyV2 and AppleBCMWLANCore integrate the original AirPort Brcm 4331 / 4360 series drivers, with more features and better logic.

New features always mean new attack surfaces.

The combination of reverse engineering and Apple SDK means a better life.

Several brand new kernel vulnerability case studies.

Q&A

wang yu

Didi Research America