black hat USA 2022

# Dive into Apple IO80211Family Vol. II

wang yu

## About me

yu.wang@cyberserval.cn

Co-founder & CEO at Cyberserval

https://www.cyberserval.com/

薮猫科技
CYBERSERVAL

## Background of this research project

Dive into Apple IO80211FamilyV2

https://www.blackhat.com/us-20/briefings/schedule/index.html#dive-into-apple-iofamilyv-20023

# The Apple 80211 Wi-Fi Subsystem

# Previously on IO80211Family

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.

From basic network communication to trusted privacy sharing between all types of Apple devices.

# Previously on IO80211Family (cont)

| | |
|---|---|
| Daemon: | airportd, sharingd … |
| Framework: | Apple80211, CoreWifi, CoreWLAN … |
| ___________________ | |
| Family drivers V2: | IO80211FamilyV2, IONetworkingFamily |
| Family drivers: | IO80211Family, IONetworkingFamily |
| Plugin drivers V2: | AppleBCMWLANCore replaces AirPort Brcm series drivers |
| Plugin drivers: | AirPortBrcmNIC, AirPortBrcm4360 / 4331, AirPortAtheros40 … |
| Low-level drivers V2: | AppleBCMWLANBusInterfacePCIe … |
| Low-level drivers: | IOPCIFamily … |

# Previously on IO80211Family (cont)

An early generation fuzzing framework, a simple code coverage analysis tool, and a Kemon-based KASAN solution.

Vulnerability classification:

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)

# Previously on IO80211Family (cont)

Some of the vulnerabilities I've introduced in detail, but others I can't disclose because they haven't been fixed before Black Hat USA 2020.

| | |
|---|---|
| Family drivers V2: | IO80211FamilyV2, IONetworkingFamily<br>CVE-2020-9832 |
| Plugin drivers V2: | AppleBCMWLANCore replaces AirPort Brcm series drivers<br>CVE-2020-9834, CVE-2020-9899, CVE-2020-10013 |
| Low-level drivers V2: | AppleBCMWLANBusInterfacePCIe …<br>CVE-2020-9833 |

# Two years have passed

All the previous vulnerabilities have been fixed, the overall security of the system has been improved. The macOS Big Sur/Monterey/Ventura has been released, and the era of Apple Silicon has arrived.

1. Apple IO80211FamilyV2 has been refactored again, and its name has been changed back to IO80211Family. What happened behind this?
2. How to identify the new attack surfaces of the 80211 Wi-Fi subsystem?
3. What else can be improved in engineering and hunting?
4. Most importantly, can we still find new high-quality kernel vulnerabilities?

# Never stop exploring

1. Change is the only constant.
2. There are always new attack surfaces, and we need to constantly accumulate domain knowledge.
3. Too many areas can be improved.
4. Yes, definitely.

# Dive into Apple IO80211Family (Again)

# Attack surface identification

I'd like to change various settings of the network while sending and receiving data.

- Traditional BSD ioctl, IOKit IOConnectCallMethod series and sysctl interfaces
- Various packet sending and receiving interfaces
- Various network setting interfaces
- Various types of network interfaces

Please Make A Dentist Appointment ASAP: Attacking IOBluetoothFamily HCI and Vendor-Specific Commands
https://www.blackhat.com/eu-20/briefings/schedule/#please-make-a-dentist-appointment-asap-attacking-iobluetoothfamily-hci-and-vendor-specific-commands-21155

# Some new cases

ifioctl()
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/bsd/net/if.c#L2854

ifioctl_nexus()
https://github.com/apple-oss-distributions/xnu/blob/main/bsd/net/if.c#L3288

skoid_create() and sysctl registration
https://github.com/apple-oss-distributions/xnu/blob/main/bsd/skywalk/core/skywalk_sysctl.c#L81

# Interfaces integration

I'd like to switch the state or working mode of the kernel state machine randomly for different network interfaces.

# ifconfig command

ap1: Access Point
awdl0: Apple Wireless Direct Link
llw0: Low-latency WLAN Interface. (Used by the Skywalk system)
utun0: Tunneling Interface
lo0: Loopback (Localhost)
gif0: Software Network Interface
stf0: 6to4 Tunnel Interface
en0: Physical Wireless
enX: Thunderbolt / iBridge / Apple T2 Controller
Bluetooth PAN / VM Network Interface
bridge0: Thunderbolt Bridge

# Domain knowledge accumulation

Read the XNU source code and documents.

Look for potential attack surface from XNU test cases:
https://github.com/apple/darwin-xnu/tree/xnu-7195.121.3/tests

# Some examples

net agent:
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/tests/netagent_race_infodisc_56244905.c
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/tests/netagent_kctl_header_infodisc_56190773.c

net bridge:
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/tests/net_bridge.c

net utun:
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/tests/net_tun_pr_35136664.c

IP6_EXTHDR_CHECK:
https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/tests/IP6_EXTHDR_CHECK_61873584.c

# Random, but not too random

So far, the new generation of Apple 80211 Wi-Fi fuzzing framework integrates more than forty network interfaces and attack surfaces.

One more thing. Is the more attack surfaces covered in each test the better?
In practice, I found that this is not the case.

# Conclusion one

- About network interfaces and attack surfaces

1. We need to accumulate as much domain knowledge as possible by learning XNU source code, documents and test cases.

2. For each round, we should randomly select two or three interface units and test them as fully as possible.

# Kernel debugging

From source code learning, static analysis to remote kernel debugging.

Make full use of LLDB and KDK:

- The information provided in the panic log is often not helpful in finding the root cause
- Variable (initial) value sometimes require dynamic analysis
- Kernel heap corruption requires remote debugging

# A kernel panic case

Without the help of the kernel debugger, there is probably no answer.

```
[(lldb) bt                                                                                                                                                                                                                  ]
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xfffffe00295de99c kernel.release.t8112`panic_trap_to_debugger [inlined] DebuggerTrapWithState(db_op=DBOP_PANIC, db_panic_str="%s %s -- exit reason namespace %d subcode 0x%llx description: %
.800s", db_panic_args=0xfffffe3d91ddf9a8, db_panic_options=32, db_panic_data_ptr=0x0000000000000000, db_proceed_on_sync_failure=1, db_panic_caller=18446741875385547616) at debug.c:715:2 [opt]
    frame #1: 0xfffffe00295de95c kernel.release.t8112`panic_trap_to_debugger(panic_format_str="%s %s -- exit reason namespace %d subcode 0x%llx description: %.800s", panic_args=0xfffffe3d91ddf9a8, reason=
0, ctx=0x0000000000000000, panic_options_mask=32, panic_data_ptr=0x0000000000000000, panic_caller=18446741875385547616) at debug.c:1176:2 [opt]
    frame #2: 0xfffffe0029dfca40 kernel.release.t8112`panic_with_options(reason=<unavailable>, ctx=<unavailable>, debugger_options_mask=<unavailable>, str=<unavailable>) at debug.c:1019:2 [opt]
    frame #3: 0xfffffe0029adbb60 kernel.release.t8112`proc_prepareexit [inlined] proc_handle_critical_exit(p=0xfffffe1666dea860, rv=10) at kern_exit.c:0 [opt]
    frame #4: 0xfffffe0029adb948 kernel.release.t8112`proc_prepareexit(p=0xfffffe1666dea860, rv=10, perf_notify=<unavailable>) at kern_exit.c:1801:3 [opt]
    frame #5: 0xfffffe0029adad14 kernel.release.t8112`exit_with_reason(p=0xfffffe1666dea860, rv=10, retval=<unavailable>, thread_can_terminate=1, perf_notify=1, jetsam_flags=0, exit_reason=0xfffffe1b3707d
8b0) at kern_exit.c:1534:2 [opt]
    frame #6: 0xfffffe0029aff834 kernel.release.t8112`postsig_locked(signum=10) at kern_sig.c:3119:3 [opt]
    frame #7: 0xfffffe0029affdbc kernel.release.t8112`bsd_ast(thread=0xfffffe1ffdbc2000) at kern_sig.c:3388:4 [opt]
    frame #8: 0xfffffe00295d6134 kernel.release.t8112`ast_taken_user at ast.c:224:3 [opt]
    frame #9: 0xfffffe002958fcb0 kernel.release.t8112`user_take_ast + 12
    frame #10: 0xfffffe002972b55c kernel.release.t8112`thread_exception_return at sleh.c:726:2 [opt]
    frame #11: 0xfffffe00295e13ec kernel.release.t8112`exception_triage_thread(exception=<unavailable>, code=<unavailable>, codeCnt=<unavailable>, thread=<unavailable>) at exception.c:698:3 [opt]
    frame #12: 0xfffffe002972d2e8 kernel.release.t8112`handle_user_abort(state=<unavailable>, esr=2181038087, fault_addr=7056804912, fault_code=<unavailable>, fault_type=<unavailable>, expected_fault_hand
ler=<unavailable>) at sleh.c:2303:2 [opt]
    frame #13: 0xfffffe002972baa8 kernel.release.t8112`sleh_synchronous(context=0xfffffe24cca2c7b0, esr=2181038087, far=7056804912) at sleh.c:0 [opt]
    frame #14: 0xfffffe002958f784 kernel.release.t8112`fleh_synchronous + 40
    frame #15: 0x00000001a49e4c30
(lldb)
```

```
| LLDB (F1) | Target (F2) | Process (F3) | Thread (F4) | View (F5) | Help (F6) |
```

```
<Sources>
kernel.release.t8112`proc_prepareexit
0xfffffe0029adb92c | cmp     x8, x19
0xfffffe0029adb930 | b.ne    -0x1fff882066c          ; <+2696> [inlined] proc_getpid at kern_proc.c:1051:12
0xfffffe0029adb934 | mov     w8, #0x0
0xfffffe0029adb938 | b       -0x1fff8820668          ; <+2700> [inlined] proc_handle_critical_exit + 80 at kern_exit.c:1748:23
0xfffffe0029adb93c | mov     w0, #0x5
0xfffffe0029adb940 | bl      -0x1fff84fa5a4          ; zone_id_require_ro_panic at zalloc.c:7005
0xfffffe0029adb944 | bl      -0x1fff8d645c8          ; __stack_chk_fail at stack_protector.c:36
0xfffffe0029adb948 |◆ldr    x9, [x8, #0x108]
0xfffffe0029adb94c | cbz     x9, -0x1fff8820688      ; <+2668> [inlined] proc_handle_critical_exit + 48 at kern_exit.c
0xfffffe0029adb950 | ldr     w10, [x8, #0x28]
0xfffffe0029adb954 | cbz     w10, -0x1fff8820688     ; <+2668> [inlined] proc_handle_critical_exit + 48 at kern_exit.c
0xfffffe0029adb958 | add     x11, x9, #0x10
0xfffffe0029adb95c | add     x10, x10, x9
0xfffffe0029adb960 | cmp     x11, x10
0xfffffe0029adb964 | b.hi    -0x1fff8820688          ; <+2668> [inlined] proc_handle_critical_exit + 48 at kern_exit.c
0xfffffe0029adb968 | ldr     w12, [x9, #0x4]
0xfffffe0029adb96c | add     x12, x11, x12
0xfffffe0029adb970 | cmp     x12, x10
0xfffffe0029adb974 | b.ls    -0x1fff88204a0          ; <+3156> [inlined] proc_handle_critical_exit + 536 at kern_exit.c
0xfffffe0029adb978 | mov     x24, #0x0
0xfffffe0029adb97c | adrp    x9, -1414
0xfffffe0029adb980 | ldr     x9, [x9, #0x620]
0xfffffe0029adb984 | cmp     x9, x19
0xfffffe0029adb988 | b.ne    -0x1fff8820648          ; <+2732> [inlined] proc_getpid at kern_proc.c:1051:12
0xfffffe0029adb98c | mov     w9, #0x0
0xfffffe0029adb990 | b       -0x1fff8820644          ; <+2736> [inlined] proc_handle_critical_exit + 116 at kern_exit.c:1751:58
0xfffffe0029adb994 | ldr     w8, [x19, #0x60]
0xfffffe0029adb998 | and     w9, w21, #0x7f
0xfffffe0029adb99c | stp     x9, x23, [sp, #0x8]
0xfffffe0029adb9a0 | str     x8, [sp]
0xfffffe0029adb9a4 | adrp    x0, -1859
0xfffffe0029adb9a8 | add     x0, x0, #0x306          ; "pid %d exited -- no exit reason available -- (signal %d, exit %d)\n"
0xfffffe0029adb9ac | bl      -0x1fff8cf6c1c          ; printf at printf.c:875
0xfffffe0029adb9b0 | mov     x24, #0x0
0xfffffe0029adb9b4 | b       -0x1fff8820620          ; <+2772> [inlined] proc_handle_critical_exit + 152 at kern_exit.c:1756:7
0xfffffe0029adb9b8 | ldr     w9, [x19, #0x60]
```

```
<Threads>
◆─process 1
 └─◆─thread #1: tid = 0x0001, stop reas
     ├─frame #0: panic_trap_to_debugger [
     ├─frame #1: panic_trap_to_debugger +
     ├─frame #2: panic_with_options + 68
     ├─frame #3: proc_prepareexit [inline
     ├─frame #4: proc_prepareexit + 2620
     ├─frame #5: exit_with_reason + 468
     ├─frame #6: postsig_locked + 1068
     ├─frame #7: bsd_ast + 1252
     ├─frame #8: ast_taken_user + 216
     ├─frame #9: user_take_ast + 12
     ├─frame #10: thread_exception_return
     ├─frame #11: exception_triage_thread
     ├─frame #12: handle_user_abort + 421
     ├─frame #13: sleh_synchronous + 1320
     ├─frame #14: fleh_synchronous + 40
     └─frame #15:
```

```
<Variables>
◆─(const char [51]) _os_log_fmt "Text page corruption detected in dying process %d\n"
◆─(proc_t) p = 0xfffffe1666dea860
(int) rv = 10
(boolean_t) perf_notify
(int) create_corpse = 0
(int) kr = 0
◆─(rusage_superset *) rup = 0x0000000000000000
◆─(thread_t) self
◆─(uthread *) ut
(exception_type_t) etype = 0
(mach_exception_data_type_t) subcode = 0
(mach_exception_data_type_t) code = 0
◆─(uintptr_t [2]) bt
◆─(backtrace_user_info) btinfo
◆─(task_t) task
(unsigned int) frame_count
```

```
Process:    1    stopped          Thread: 0x0001      Frame:   4  PC = 0xfffffe0029adb948
```

# Kernel Debug Kit

*"Note: Apple silicon doesn't support active kernel debugging. … you cannot set breakpoints, continue code execution, step into code, step over code, or step out of the current instruction."*

Asahi Linux
https://asahilinux.org/

An Overview of macOS Kernel Debugging
https://blog.quarkslab.com/an-overview-of-macos-kernel-debugging.html

LLDBagility: Practical macOS Kernel Debugging
https://blog.quarkslab.com/lldbagility-practical-macos-kernel-debugging.html

# Conclusion two

- About network interfaces and attack surfaces

- About static and dynamic analysis methods

1. We should make full use of LLDB kernel debugging environment, KDK and public symbols for reverse engineering.

2. At this stage, we need the help of third-party solutions for the Apple Silicon platform.

# Kernel Address Sanitizer

The previous panic is a typical case of corruption, and we need help from KASAN.

However, we need to do some fixes because sometimes the built-in tools/kernels don't work very well.

We even need to implement KASAN-like solution to dynamically monitor special features of third-party kernel extensions.

# An obstacle case

console_io_allowed()

https://github.com/apple/darwin-xnu/blob/xnu-7195.121.3/osfmk/console/serial_console.c#L162

```
static inline bool
console_io_allowed(void)
{
    if (!allow_printf_from_interrupts_disabled_context &&
        !console_suspended &&
        startup_phase >= STARTUP_SUB_EARLY_BOOT &&
        !ml_get_interrupts_enabled()) {
#if defined(__arm__) || defined(__arm64__) || DEBUG || DEVELOPMENT
        panic("Console I/O from interrupt-disabled context");
#else
        return false;
#endif
    }

    return true;
}
```

```
Process 1 stopped
* thread #1, stop reason = signal SIGSTOP
    frame #0: 0xffffff800e4d82da kernel.kasan`DebuggerTrapWithState [inlined] current_debugger_state at debug.c:176:9 [opt]
Target 0: (kernel.kasan) stopped.
(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff800e4d82da kernel.kasan`DebuggerTrapWithState [inlined] current_debugger_state at debug.c:176:9 [opt]
    frame #1: 0xffffff800e4d82da kernel.kasan`DebuggerTrapWithState(db_op=DBOP_PANIC, db_message="panic", db_panic_str="\"Console I/O from interrupt-disabled context\"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6
c82cc8/Library/Caches/com.apple.xbs/Sources/xnu_kasan/xnu-7195.141.26/osfmk/console/serial_console.c:162", db_panic_args=0xffffffb0c2a0ee30, db_panic_options=0, db_panic_data_ptr=0x0000000000000000, db_proceed_on_s
ync_failure=1, db_panic_caller=18446743524197246046) at debug.c:598:8 [opt]
    frame #2: 0xffffff800e4d9041 kernel.kasan`panic_trap_to_debugger(panic_format_str="\"Console I/O from interrupt-disabled context\"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6c82cc8/Library/Caches/com.apple.x
bs/Sources/xnu_kasan/xnu-7195.141.26/osfmk/console/serial_console.c:162", panic_args=<unavailable>, reason=0, ctx=0x0000000000000000, panic_options_mask=0, panic_data_ptr=<unavailable>, panic_caller=184467435241972
46046) at debug.c:938:2 [opt]
    frame #3: 0xffffff800fc5ed23 kernel.kasan`panic(str=<unavailable>) at debug.c:803:2 [opt]
    frame #4: 0xffffff800e83a45e kernel.kasan`console_write [inlined] console_io_allowed at serial_console.c:162:3 [opt]
    frame #5: 0xffffff800e83a411 kernel.kasan`console_write(str="ethernet MAC address: 60:f8:1d:b4:60:36\n", size=0) at serial_console.c:451:6 [opt]
    frame #6: 0xffffff800e839756 kernel.kasan`console_printbuf_putc(ch=10, arg=0xffffffb0c2a0eff8) at serial_general.c:184:3 [opt]
    frame #7: 0xffffff800e517669 kernel.kasan`__doprnt(fmt="\n", argp=0xffffffb0c2a0efe0, putc=(kernel.kasan`console_printbuf_putc at serial_general.c:160), arg=0xffffffb0c2a0eff8, radix=16, is_log=1) at printf.c:0
:2 [opt]
    frame #8: 0xffffff800e518cb5 kernel.kasan`vprintf_internal(fmt="ethernet MAC address: %02x:%02x:%02x:%02x:%02x:%02x\n", ap_in=0xffffffb0c2a0f170, caller=0xffffff800e45a99b) at printf.c:915:4 [opt]
    frame #9: 0xffffff800e518b55 kernel.kasan`printf(fmt=<unavailable>) at printf.c:938:8 [opt]
    frame #10: 0xffffff800e45a99b kernel.kasan`kdp_raise_exception [inlined] kdp_connection_wait at kdp_udp.c:1214:3 [opt]
    frame #11: 0xffffff800e45a908 kernel.kasan`kdp_raise_exception [inlined] kdp_debugger_loop(exception=<unavailable>, code=<unavailable>, subcode=<unavailable>, saved_state=0xffffffb0c2a0f3a0) at kdp_udp.c:1426:3
 [opt]
    frame #12: 0xffffff800e45a495 kernel.kasan`kdp_raise_exception(exception=<unavailable>, code=3, subcode=0, saved_state=0xffffffb0c2a0f3a0) at kdp_udp.c:2404:2 [opt]
    frame #13: 0xffffff800e4d881e kernel.kasan`handle_debugger_trap(exception=6, code=<unavailable>, subcode=0, state=0xffffffb0c2a0f3a0) at debug.c:1285:3 [opt]
    frame #14: 0xffffff800e8b854f kernel.kasan`kdp_i386_trap(trapno=<unavailable>, saved_state=0xffffffb0c2a0f3a0, result=<unavailable>, va=140462291755008) at kdp_machdep.c:441:2 [opt]
    frame #15: 0xffffff800e8a2a63 kernel.kasan`kernel_trap(state=0xffffffb0c2a0f390, lo_spp=<unavailable>) at trap.c:774:7 [opt]
    frame #16: 0xffffff800e8c091f kernel.kasan`trap_from_kernel + 38
    frame #17: 0xffffff800e8b83e5 kernel.kasan`kdp_call at kdp_machdep.c:338:1 [opt]
    frame #18: 0xffffff800e45834c kernel.kasan`kdp_set_ip_and_mac_addresses [inlined] debugger_if_necessary at kdp_udp.c:692:3 [opt]
    frame #19: 0xffffff800e45832f kernel.kasan`kdp_set_ip_and_mac_addresses(ipaddr=<unavailable>, macaddr=<unavailable>) at kdp_udp.c:800:2 [opt]
    frame #20: 0xffffff800edb9be9 kernel.kasan`ether_inet_prmod_ioctl(ifp=<unavailable>, protocol_family=<unavailable>, command=<unavailable>, data=<unavailable>) at ether_inet_pr_module.c:360:4 [opt]
    frame #21: 0xffffff800ed9261b kernel.kasan`ifnet_ioctl(ifp=<unavailable>, proto_fam=<unavailable>, ioctl_code=<unavailable>, ioctl_arg=0xffffff8ad560b230) at dlil.c:7224:14 [opt]
    frame #22: 0xffffff800f07f360 kernel.kasan`in_ifinit(ifp=<unavailable>, ia=<unavailable>, sin=0xffffff8ad53641fe, scrub=<unavailable>) at in.c:1779:10 [opt]
    frame #23: 0xffffff800f07b076 kernel.kasan`inctl_ifaddr(ifp=0xffffff8ad5364140, ia=0xffffff8ad560b230, cmd=<unavailable>, ifr=0xffffffb000000000) at in.c:730:12 [opt]
    frame #24: 0xffffff800f0765c4 kernel.kasan`in_control(so=<unavailable>, cmd=2151704858, data="en0", ifp=0xffffff8ad5364140, p=<unavailable>) at in.c:1580:11 [opt]
    frame #25: 0xffffff800ed6ba9f kernel.kasan`ifioctl(so=0xffffff8ad9ad9be0, cmd=2151704858, data="en0", p=<unavailable>) at if.c:3302:12 [opt]
    frame #26: 0xffffff800ed7453f kernel.kasan`ifioctllocked(so=0xffffff8ad9ad9be0, cmd=<unavailable>, data=<unavailable>, p=<unavailable>) at if.c:4186:10 [opt]
    frame #27: 0xffffff800f54ba55 kernel.kasan`soioctl(so=<unavailable>, cmd=<unavailable>, data=<unavailable>, p=<unavailable>) at sys_socket.c:266:11 [opt]
    frame #28: 0xffffff800f425eb3 kernel.kasan`fo_ioctl(fp=0xffffff8ad7990180, com=2151704858, data=<unavailable>, ctx=0xffffffb0c2a0fdd0) at kern_descrip.c:5662:10 [opt]
    frame #29: 0xffffff800f53be44 kernel.kasan`ioctl(p=<unavailable>, uap=<unavailable>, retval=<unavailable>) at sys_generic.c:1067:11 [opt]
    frame #30: 0xffffff800f8b2910 kernel.kasan`unix_syscall64(state=0xffffff8ad7c43310) at systemcalls.c:412:10 [opt]
    frame #31: 0xffffff800e8c10e6 kernel.kasan`hndl_unix_scall64 + 22
(lldb) continue
Process 1 resuming
(lldb)
```

# KASAN and code coverage analysis

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

I have ported Kemon and the kernel inline engine to the Apple Silicon platform.

# Conclusion three

- About network interfaces and attack surfaces
- About static and dynamic analysis methods
- About creating tools

1. We need to do fixes because sometimes the built-in tools don't work very well.
2. We even need to implement KASAN-like solution, code coverage analysis tool to dynamically monitor third-party closed source kernel extensions.

# Apple SDKs and build-in tools

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

Build-in network and Wi-Fi tools

# 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
```

# Conclusion

- About network interfaces and attack surfaces
- About static and dynamic analysis methods
- About creating tools
- About others

1. Pay attention to the tools provided in the macOS/iOS operating system.
2. We should make full use of the apple SDKs, and contribute to Wi-Fi developer community.

# DEMO

Apple 80211 Wi-Fi subsystem fuzzing framework
on the latest macOS Ventura 13.0 Beta 4 (22A5311f)

# Apple 80211 Wi-Fi Subsystem
# Latest Zero-day Vulnerability Case Studies

# Follow-up ID and CVE ID

Apple Product Security Follow-up IDs:
791541097 (CVE-2022-32837), 797421595 (CVE-2022-26761), 797590499 (CVE-2022-26762), OE089684257715 (CVE-2022-32860), OE089692707433 (CVE-2022-32847), OE089712553931, OE089712773100, OE0900967233115, OE0908765113017, OE090916270706, etc.

CVE-2020-9899:
AirPortBrcmNIC`AirPort_BrcmNIC::setROAM_PROFILE
Kernel Stack Overflow Vulnerability

About the security content of macOS Catalina 10.15.6,
Security Update 2020-004 Mojave,
Security Update 2020-004 High Sierra
https://support.apple.com/en-us/HT211289

Two years have passed, are there still such high-quality arbitrary (kernel) memory write vulnerabilities?

# Yes, definitely

CVE-2022-32847:
AirPort_BrcmNIC::setup_btc_select_profile
Kernel Stack Overwrite Vulnerability

About the security content of iOS 15.6 and iPadOS 15.6
https://support.apple.com/en-us/HT213346

About the security content of macOS Monterey 12.5
https://support.apple.com/en-us/HT213345

About the security content of macOS Big Sur 11.6.8
https://support.apple.com/en-us/HT213344

```
Process 1 stopped
* thread #1, stop reason = EXC_BAD_ACCESS (code=10, address=0xd1dd0000)
    frame #0: 0xffffff8005a53fbb
->  0xffffff8005a53fbb: cmpl   $0x1, 0x18(%rbx,%rcx,4)
    0xffffff8005a53fc0: cmovnel %esi, %edi
    0xffffff8005a53fc3: orl    %edi, %edx
    0xffffff8005a53fc5: incq   %rcx
Target 0: (kernel.kasan) stopped.
(lldb) register read
General Purpose Registers:
       rax = 0x00000000481b8d16
       rbx = 0xffffffb0d1dcf3f4
       rcx = 0x00000000000002fd
       rbp = 0xffffffb0d1dcf3e0
       rsp = 0xffffffb0d1dcf3c0
       rip = 0xffffff8005a53fbb  AirPortBrcmNIC`AirPort_BrcmNIC::setup_btc_select_profile + 61
(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff8005a53fbb AirPortBrcmNIC`AirPort_BrcmNIC::setup_btc_select_profile + 61
```

CVE-2020-10013:
AppleBCMWLANCoreDbg
Arbitrary Memory Write Vulnerability

About the security content of iOS 14.0 and iPadOS 14.0
https://support.apple.com/en-us/HT211850

About the security content of macOS Catalina 10.15.7,
Security Update 2020-005 High Sierra,
Security Update 2020-005 Mojave
https://support.apple.com/en-us/HT211849

black hat USA 2022

```
kernel`bcopy:
->  0xffffff8000398082  <+18>: rep
    0xffffff8000398083  <+19>: movsb  (%rsi), %es:(%rdi)
    0xffffff8000398084  <+20>: retq
    0xffffff8000398085  <+21>: addq   %rcx, %rdi
(lldb) register read rcx rsi rdi
General Purpose Registers:
       rcx = 0x0000000000000023
       rsi = 0xffffff81b1d5e000
       rdi = 0xffffff80deadbeef
(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff8000398082 kernel`bcopy + 18
    frame #1: 0xffffff800063abd4 kernel`memmove + 20
    frame #2: 0xffffff7f828e1a64 AppleBCMWLANCore`AppleBCMWLANUserPrint + 260
    frame #3: 0xffffff7f8292bab7 AppleBCMWLANCore`AppleBCMWLANCoreDbg::cmdSetScanIterationTimeout + 91
    frame #4: 0xffffff7f82925949 AppleBCMWLANCore`AppleBCMWLANCoreDbg::dispatchCommand + 479
    frame #5: 0xffffff7f828b37bd AppleBCMWLANCore::apple80211Request + 1319
```

# Summary of case #3

1. CVE-2020-10013 is an arbitrary memory write vulnerability caused by boundary checking errors.

2. The value to be written is predictable or controllable.

3. Combined with kernel information disclosure vulnerabilities, a complete local EoP exploit chain can be formed. The write primitive is stable and does not require heap Feng Shui manipulation.
   CVE-2020-9833 (p44-p49):
   https://i.blackhat.com/USA-20/Thursday/us-20-Wang-Dive-into-Apple-IO80211FamilyV2.pdf

4. This vulnerability affects hundreds of AppleBCMWLANCoreDbg handlers!

Two years have passed, are there still such high-quality arbitrary (kernel) memory write vulnerabilities?

# Yes, definitely

CVE-2022-26762:
IO80211Family`getRxRate
Arbitrary Memory Write Vulnerability

About the security content of iOS 15.5 and iPadOS 15.5
https://support.apple.com/en-us/HT213258

About the security content of macOS Monterey 12.4
https://support.apple.com/en-us/HT213257

```
Process 1 stopped
* thread #1, stop reason = signal SIGSTOP
    frame #0: 0xffffff8008b23ed7 IO80211Family`getRxRate + 166
IO80211Family`getRxRate:
->  0xffffff8008b23ed7 <+166>: movl   %eax, (%rbx)
    0xffffff8008b23ed9 <+168>: xorl   %eax, %eax
    0xffffff8008b23edb <+170>: movq   0xca256(%rip), %rcx
    0xffffff8008b23ee2 <+177>: movq   (%rcx), %rcx
Target 2: (kernel) stopped.
(lldb) register read
General Purpose Registers:
       rax = 0x0000000000000258
       rbx = 0xdeadbeefdeadcafe
       rip = 0xffffff8008b23ed7  IO80211Family`getRxRate + 166
(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff8008b23ed7 IO80211Family`getRxRate + 166
    frame #1: 0xffffff8008af9326 IO80211Family`IO80211Controller::_apple80211_ioctl_getLegacy + 70
    frame #2: 0xffffff8008b14adc IO80211Family`IO80211SkywalkInterface::performGatedCommandIOCTL + 274
```

# Summary of case #4

1. Compared with CVE-2020-10013, the root cause of CVE-2022-26762 is simpler: the vulnerable kernel function forgets to sanitize user-mode pointer. These simple and stable kernel Vulnerabilities are powerful, they are perfect for Pwn2Own.

2. The value to be written is fixed. The write primitive is stable and does not require heap Feng Shui manipulation.

3. Kernel vulnerabilities caused by copyin/copyout, copy_from_user/copy_to_user, ProbeForRead/ProbeForWrite are very common.

4. All inputs are potentially harmful.

CVE-2022-32860 and CVE-2022-32837
Kernel Out-of-bounds Read and Write Vulnerability

About the security content of iOS 15.6 and iPadOS 15.6
https://support.apple.com/en-us/HT213346

About the security content of macOS Monterey 12.5
https://support.apple.com/en-us/HT213345

About the security content of macOS Big Sur 11.6.8
https://support.apple.com/en-us/HT213344

```
Kernel slid 0x16e10000 in memory.
Loaded kernel file /Library/Developer/KDKs/KDK_11.6.5_20G527.kdk/System/Library/Kernels/kernel.kasan
warning: 'kernel' contains a debug script. To run this script in this debug session:

    command script import "/Library/Developer/KDKs/KDK_11.6.5_20G527.kdk/System/Library/Kernels/kernel.kasan.dSYM/Contents/Resources/Python/kernel.py"

To run all discovered debug scripts in this session:

    settings set target.load-script-from-symbol-file true

Loading 155 kext modules -----.--------------...--------------.-----.----------.------------------------------.------------..------------.--------------------.------------------.- done.
Process 1 stopped
* thread #1, stop reason = signal SIGSTOP
    frame #0: 0xffffff80170d82da kernel.kasan`DebuggerTrapWithState [inlined] current_debugger_state at debug.c:176:9 [opt]
Target 13: (kernel.kasan) stopped.
(lldb) register read
General Purpose Registers:
       rax = 0x0000000000000001
       rbx = 0x0000000000000000
       rcx = 0x0000000000000000
       rdx = 0xffffff8018946c39  ""%s"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6c82cc8/Library/Caches/com.apple.xbs/Sources/xnu_kasan/xnu-7195.141.26/san/kasan.c:511"
       rdi = 0x0000000000000003
       rsi = 0xffffff801888163a  "panic"
       rbp = 0xffffffa078faec80
       rsp = 0xffffffa078faec40
        r8 = 0x0000000000000000
        r9 = 0x0000000000000000
       r10 = 0x0000000000000000
       r11 = 0x0000000000000008
       r12 = 0x0000000000000000
       r13 = 0xffffff80197e6680  kernel.kasan`percpu_slot_debugger_state
       r14 = 0xffffff8018946c39  ""%s"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6c82cc8/Library/Caches/com.apple.xbs/Sources/xnu_kasan/xnu-7195.141.26/san/kasan.c:511"
       r15 = 0x0000000000000003
       rip = 0xffffff80170d82da  kernel.kasan`DebuggerTrapWithState + 202 [inlined] current_debugger_state at debug.c:176:9
  kernel.kasan`DebuggerTrapWithState + 202 at debug.c:598:8
    rflags = 0x0000000000000046
        cs = 0x0000000000000008
        fs = 0x000000001fff0000
        gs = 0x000000000f1f0000

(lldb) bt
* thread #1, stop reason = signal SIGSTOP
  * frame #0: 0xffffff80170d82da kernel.kasan`DebuggerTrapWithState [inlined] current_debugger_state at debug.c:176:9 [opt]
    frame #1: 0xffffff80170d82da kernel.kasan`DebuggerTrapWithState(db_op=DBOP_PANIC, db_message="panic", db_panic_str="\"%s\"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6c82cc8/Library/Caches/com.apple.xbs/Sourc
es/xnu_kasan/xnu-7195.141.26/san/kasan.c:511", db_panic_args=0xffffffa078faed30, db_panic_options=0, db_panic_data_ptr=0x0000000000000000, db_proceed_on_sync_failure=1, db_panic_caller=18446743524365276617) at debu
g.c:598:8 [opt]
    frame #2: 0xffffff80170d9041 kernel.kasan`panic_trap_to_debugger(panic_format_str="\"%s\"@/System/Volumes/Data/SWE/macOS/BuildRoots/a0c6c82cc8/Library/Caches/com.apple.xbs/Sources/xnu_kasan/xnu-7195.141.26/san/
kasan.c:511", panic_args=<unavailable>, reason=0, ctx=0x0000000000000000, panic_options_mask=0, panic_data_ptr=<unavailable>, panic_caller=18446743524365276617) at debug.c:938:2 [opt]
    frame #3: 0xffffff801885ed23 kernel.kasan`panic(str=<unavailable>) at debug.c:803:2 [opt]
    frame #4: 0xffffff80188795c9 kernel.kasan`kasan_report_internal.cold.1 at kasan.c:511:3 [opt]
    frame #5: 0xffffff80188533bd kernel.kasan`kasan_report_internal(p=<unavailable>, width=<unavailable>, access=<unavailable>, reason=<unavailable>, dopanic=<unavailable>) at kasan.c:511:3 [opt]
    frame #6: 0xffffff8018851533 kernel.kasan`kasan_crash_report(p=<unavailable>, width=<unavailable>, access=<unavailable>, reason=<unavailable>) at kasan.c:521:2 [opt]
    frame #7: 0xffffff801885101a kernel.kasan`kasan_violation(addr=<unavailable>, size=<unavailable>, access=<unavailable>, reason=<unavailable>) at kasan.c:279:2 [opt]
    frame #8: 0xffffff80188521f9 kernel.kasan`kasan_check_free(addr=18446743662184594176, size=256, heap_type=1) at kasan.c:1126:3 [opt]
    frame #9: 0xffffff80170ff5e9 kernel.kasan`kfree_ext(kheap=<unavailable>, data=<unavailable>, size=256) at kalloc.c:1118:2 [opt]
    frame #10: 0xffffff80170ff3b8 kernel.kasan`kfree(addr=<unavailable>, size=<unavailable>) at kalloc.c:1162:2 [opt] [artificial]
    frame #11: 0xffffff801860e0f4 kernel.kasan`::IOFree(inAddress=<unavailable>, size=256) at IOLib.cpp:374:3 [opt]
    frame #12: 0xffffff801a633fbd AirPortBrcmNIC`osl_mfree + 330
    frame #13: 0xffffff801a66198e AirPortBrcmNIC`AirPort_BrcmNIC::setAWDL_SYNCHRONIZATION_CHANNEL_SEQUENCE(OSObject*, apple80211_awdl_sync_channel_sequence*) + 742
    frame #14: 0xffffff801a6708e0 AirPortBrcmNIC`AirPort_BrcmNIC::apple80211VirtualRequest(unsigned int, int, IO80211VirtualInterface*, void*) + 3072
```

CVE-2022-26761:
IO80211AWDLPeerManager::updateBroadcastMI
Out-of-bounds Read and Write Vulnerability caused by Type Confusion

About the security content of macOS Monterey 12.4
https://support.apple.com/en-us/HT213257

About the security content of macOS Big Sur 11.6.6
https://support.apple.com/en-us/HT213256

# Takeaways and The End

# From the perspective of kernel development

1. Apple has made a lot of efforts, and the security of macOS/iOS has been significantly improved.
2. All inputs are potentially harmful, kernel developers should carefully check all input parameters.
3. New features always mean new attack surfaces.
4. Callback functions, especially those that support different architectures or working modes, and state machine, exception handling need to be carefully designed.
5. Corner cases matter.

# From the perspective of vulnerability research

1. Arbitrary kernel memory write vulnerabilities represented by CVE-2022-26762 are powerful, they are simple and stable enough.

2. Combined with kernel information disclosure vulnerabilities such as CVE-2020-9833, a complete local EoP exploit chain can be formed.

3. Stack out-of-bounds read and write vulnerabilities represented by CVE-2022-32847 are often found. The root cause is related to stack-based variables being passed and used for calculation or parsing. The stack canary can't solve all the problems.

# From the perspective of vulnerability research (cont)

4. Vulnerabilities represented by CVE-2022-26761 indicate that handlers that support different architectures or working modes are prone to problems.

5. Vulnerabilities represented by CVE-2020-9834 and Follow-up ID OE0908765113017 indicate that some handlers with complex logic will be introduced with new vulnerabilities every once in a while, even if the old ones have just been fixed.

# From the perspective of engineering and hunting

1. It is important to integrate subsystem interfaces at different levels and their attack surfaces.
2. It is important to integrate KASAN and code coverage analysis tools.
3. Many work needs to be ported to Apple Silicon platform, such as Kemon.
4. We should combine all available means such as reverse engineering, kernel debugging, XNU resources, Apple SDKs, third-party tools, etc.
5. If you've done this, or just started, you'll find that Apple did a lot of work, but the results seem to be similar to 2020.

# Q&A

wang yu

Cyberserval