BY

Blue
Frost
Security

Unearthing Vulnerabilities in the Apple Ecosystem: The Art of KidFuzzerV2.0

Pan Zhenpeng(@Peterpan0927)

STAR LABS SG Pte. Ltd.

OFFENSIVE CON

Bio

- Pan Zhenpeng @peterpan980927 on Twitter
- Security Researcher of STAR Labs SG Pte. Ltd.
- Focus on iOS/macOS/Web bug hunting and exploit
- Speaker of Zer0Con2021 and POC2022
- Previously working at Alibaba Security/Qihoo 360

Agenda

- Backgrounds
- KidFuzzerV2 & bug analysis
- Summary

What could you learn from this talk?

- Background knowledge about Apple Ecosystem
  - Structures overview
  - Attack surfaces overview
  - Mitigations & VR thoughts
- How to develop a simple but effective fuzzer from scratch
- The "golden ticket" towards some easy Apple 0day CVEs

Backgrounds

Apple Kernel Space Structure

- XNU – Hybrid Kernel
  - Mach
  - BSD
  - IOKit
- Co-processors
  - aop, dcp, sio, pmp, smc, gfx-asc, sep, ans, scaler, ave, pmgr, etc

Apple Kernel Space Structure

- XNU – Hybrid Kernel
  - Mach
    - Micro kernel
    - Mach ports, a general wrapper for kobjects and mq, carry raw data/rights/memory/…
    - Task, thread, voucher…
  - BSD
  - IOKit
- Co-processors

Apple Kernel Space Structure

- XNU – Hybrid Kernel
  - Mach
  - BSD
    - Monolithic kernel
    - Traditional *nix struct, fd, socket, device drivers
  - IOKit
- Co-processors

Apple Kernel Space Structure

- XNU – Hybrid Kernel
  - Mach
  - BSD
  - IOKit
    - C++ IOKit framework (IOService/IODMACommand/…)
    - Tool Drivers (IOSurface/Framebuffer/*Family/…)
    - ~~Network Filter, Endpoint Security~~
    - Peripheral drivers (AppleM2ScalerCSCDriver/…)
- Co-processors

Apple Kernel Space Structure

- XNU – Hybrid Kernel
  - Mach
  - BSD
  - IOKit
    - C++ IOKit framework (IOService/IODMACommand/…)
    - Tool Drivers (IOSurface/Framebuffer/*Family/…)
    - ~~Network Filter, Endpoint Security~~
    - Peripheral drivers (AppleM2ScalerCSCDriver/…) <---> Co-processors
- Co-processors

Apple Kernel Space Structure

- XNU
- Co-processors (direct)

```
+-o scaler0/1@B000000
    +-o AppleM2ScalerCSCDriver
…
+-o ave0/1@D100000
    +-o AppleAVE2Driver
```

Apple Kernel Space Structure

- XNU
- Co-processors (single endpoint)

```
 +-o pmp@8EC00000
 | +-o AppleASCWrapV4(com.apple.driver.AppleA7IOP/Mailbox)
 |   +-o iop-pmp-nub
 |     +-o RTBuddyV2
 |       +-o PMPEndpoint1
 |         +-o ApplePMPv2
```

Apple Kernel Space Structure

- XNU
- Co-processors (multi endpoints)

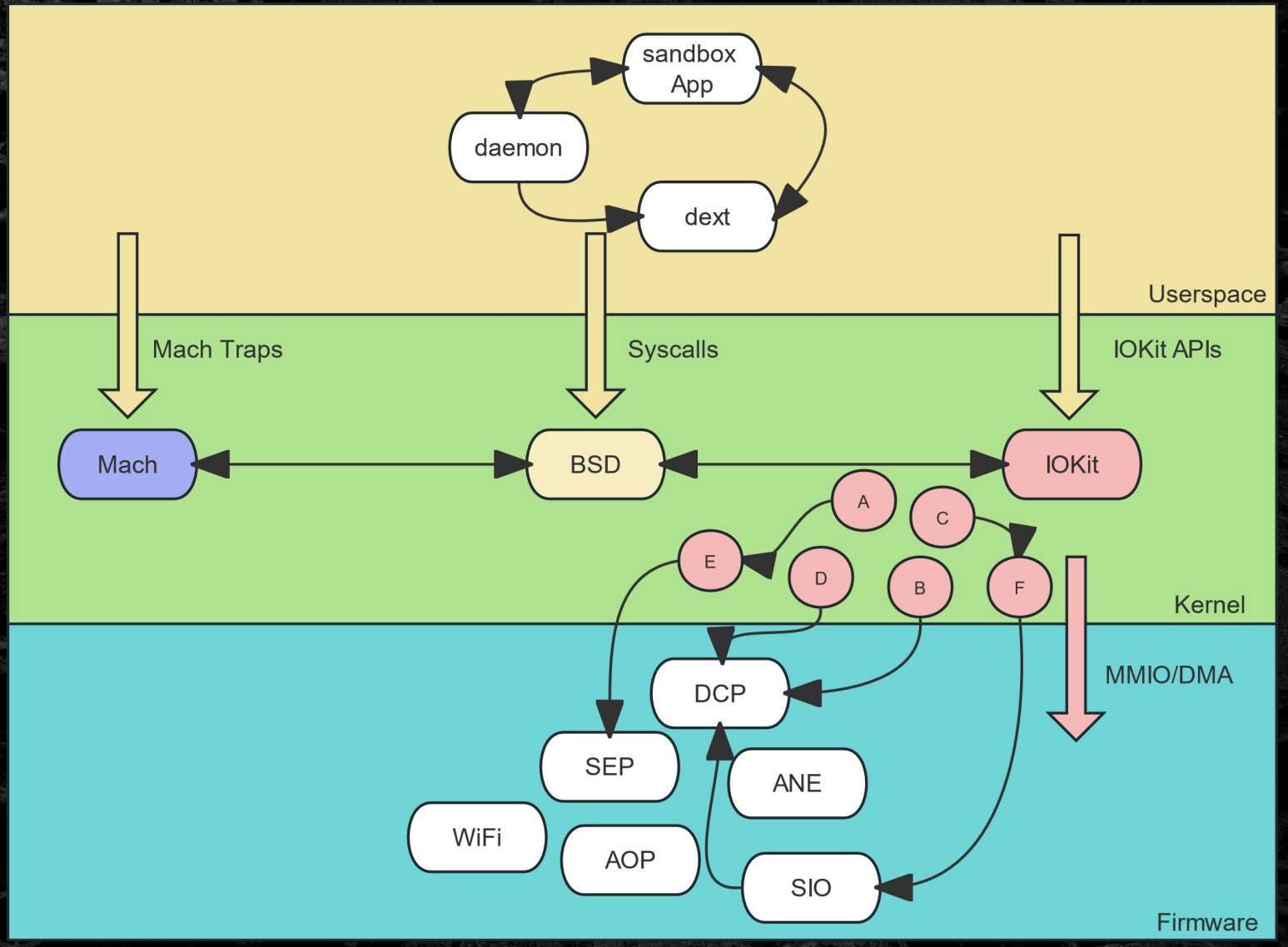
```
 +-o sep@96400000
 | +-o AppleASCWrapV4SEP(com.apple.driver.AppleA7IOP/Mailbox)
 |   +-o iop-sep-nub
 |     +-o AppleSEPManager
 |         +-o sep-endpoint, sks (AppleSEPKeyStore)
 |         +-o sep-endpoint, sbio (AppleBiometricService)
 |         +-o sep-endpoint, ...
```

Apple Kernel Space Structure

- XNU
- Co-processors (tool)

```
+-o sio@9BC00000
| +-o AppleASCWrapV4(com.apple.driver.AppleA7IOP/Mailbox)
|   +-o iop-sio-nub
|     +-o RTBuddyV2
|       +-o SIOEndpoint1
|         +-o AppleSmartIO
|           +-o sio-dma(AppleSmartIODMANub)
|             +-o IODMAController00000098 (AppleSmartIODMAController)
```

Apple Kernel Space Structure



OFFENSIVE

Apple Kernel Space Attack Surfaces

- XNU
  - Mach Traps
  - BSD Syscalls
  - IOKit APIs
- Co-processors
  - Remote attacks (wifi, baseband, bluetooth…)
  - Local attacks (DCP, SEP, GPU…)

Apple Kernel Space Attack Surfaces

- XNU
  - Mach Traps
  - BSD Syscalls
  - IOKit APIs
- Co-processors
  - Remote attacks (wifi, baseband, bluetooth…)
  - Local attacks (DCP, SEP, GPU…)
    - DMA
    - MMIO (Mailbox)

Apple Kernel Space Mitigations

- Strong and significant mitigations in XNU 😭
  - SAD FENGSHUI
  - VA SEQUESTER
  - Kalloc_type(...)
  - PPL/PAC/KTRR
  - Keep isolating user controlled memory(KMEM_RANGE_ID_SPRAYQTN/…)

Apple Kernel Space Mitigations

- Strong and significant mitigations in XNU 😭
  - SAD FENGSHUI
  - VA SEQUESTER
  - Kalloc_type(...)
  - PPL/PAC/KTRR
  - Keep isolating user controlled memory(KMEM_RANGE_ID_SPRAYQTN/...)
- Almost no modern software mitigation in Co-processors 😍
  - SCIP (Co-processor side KTRR)/PPL
  - IODART (Apple SMMU), protect attacks from the co-processors, but kernel is very likely to trust the data from co-processors, it won't be hard to cause other bugs on AP side.

Apple Kernel Space VR thoughts

- Why didn’t people all target at Co-processors?
  - Limited public resources
  - Heavy reverse engineering work 🥲
  - Hard to debug or determine runtime memory layout

Apple Kernel Space VR thoughts

- But Co-processors could be really valuable targets in the future under the modern mitigations in XNU
  - Limited public resources also means possibilities
  - Reverse engineering work is only about time
  - Emulate firmwares to solve debug problems

KidFuzzerV2 & bug analysis

OFFENSIVECON

Quick review of KidFuzzerV1

- early_fuzz
  - Collect the entitlements needed by driver, sign the fuzzer
  - Collect all reachable io-services and userclients
  - Start early_fuzz and collect effective data (maybe with root priv 😈)
  - Use the data to generate a plist for deep_fuzz stage
  - Other trivial operations, rank potential vulnerable kexts, etc

Differences in KidFuzzerV2

- early_fuzz
  - Collect the entitlements needed by driver, sign the fuzzer
  - Collect all reachable io-services and userclients
  - Start early_fuzz and collect effective data (maybe with root priv 😈)
  - Use the data to generate a plist for deep_fuzz stage
  - Other trivial operations, rank potential vulnerable kexts, etc

Entitlements collection

- Collect all Code_Near_Call xrefs to the entitlement "getters", e.g:

```
ADRL        X1, aComAppleAopRos ; entitlement
MOV         X0, X22 ; task
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Find Data_Offset xrefs from ADRL instruction to get the string addr
- Handle some special cases such as two layers xrefs:

```
MOV         X0, X1  ; task
MOV         X1, X2  ; entitlement
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Get entitlemens string from xref.to with utf-8 encoding

Entitlements collection

- Collect all Code_Near_Call xrefs to the entitlement "getters", e.g:

```
ADRL        X1, aComAppleAopRos ; entitlement
MOV         X0, X22 ; task
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Find Data_Offset xrefs from ADRL instruction to get the string addr
- Handle some special cases such as two layers xrefs:

```
MOV         X0, X1  ; task
MOV         X1, X2  ; entitlement
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Get entitlemens string from xref.to with utf-8 encoding

Entitlements collection

- Collect all Code_Near_Call xrefs to the entitlement "getters", e.g:

```
ADRL        X1, aComAppleAopRos ; entitlement
MOV         X0, X22 ; task
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Find Data_Offset xrefs from ADRL instruction to get the string addr
- Handle some special cases such as two layers xrefs:

```
MOV         X0, X1  ; task
MOV         X1, X2  ; entitlement
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Get entitlemens string from xref.to with utf-8 encoding

Entitlements collection

- Collect all Code_Near_Call xrefs to the entitlement "getters", e.g:

```
ADRL        X1, aComAppleAopRos ; entitlement
MOV         X0, X22 ; task
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Find Data_Offset xrefs from ADRL instruction to get the string addr
- Handle some special cases such as two layers xrefs:

```
MOV         X0, X1  ; task
MOV         X1, X2  ; entitlement
BL          __ZN12IOUserClient21copyClientEntitlementEP4taskPKc
```

- Get entitlemens string from xref.to with utf-8 encoding (177 ents collected)

UserClients collection

// IOKitDiagnostics = {Instance allocation=0xffa327,Container allocation=0x7b7f8e,Pageable allocation=0xe15d0000, Classes={…}, IOMalloc allocation=0x8a7425d}

```
IOMasterPort(MACH_PORT_NULL, &master_port);
root_entry = IORegistryGetRootEntry(master_port);
IORegistryEntryCreateCFProperties(root_entry, &properties, kCFAllocatorDefault, kNilOptions);

// get IOKitDiagnostics
diagnostics = CFDictionaryGetValue(properties, CFSTR(kIOKitDiagnosticsKey));
classes = (CFDictionaryRef)CFDictionaryGetValue(diagnostics, CFSTR("Classes"));
CFDictionaryApplyFunction(classes, collect_service, NULL);
```

UserClients collection

```
//  IOKitDiagnostics = {Instance allocation=0xffa327,Container allocation=0x7b7f8e,Pageable allocation=0xe15d0000, Classes={…}, IOMalloc allocation=0x8a7425d}

IOMasterPort(MACH_PORT_NULL, &master_port);
root_entry = IORegistryGetRootEntry(master_port);
IORegistryEntryCreateCFProperties(root_entry, &properties, kCFAllocatorDefault, kNilOptions);

// get IOKitDiagnostics
diagnostics = CFDictionaryGetValue(properties, CFSTR(kIOKitDiagnosticsKey));
classes = (CFDictionaryRef)CFDictionaryGetValue(diagnostics, CFSTR("Classes"));
CFDictionaryApplyFunction(classes, collect_service, NULL);
```

UserClients collection

```
//  IOKitDiagnostics = {Instance allocation=0xffa327,Container allocation=0x7b7f8e,Pageable allocation=0xe15d0000, Classes={…}, IOMalloc allocation=0x8a7425d}

IOMasterPort(MACH_PORT_NULL, &master_port);
root_entry = IORegistryGetRootEntry(master_port);
IORegistryEntryCreateCFProperties(root_entry, &properties, kCFAllocatorDefault, kNilOptions);

// get IOKitDiagnostics
diagnostics = CFDictionaryGetValue(properties, CFSTR(kIOKitDiagnosticsKey));
classes = (CFDictionaryRef)CFDictionaryGetValue(diagnostics, CFSTR("Classes"));
CFDictionaryApplyFunction(classes, collect_service, NULL);
```

UserClients collection

```
//  IOKitDiagnostics = {Instance allocation=0xffa327,Container allocation=0x7b7f8e,Pageable allocation=0xe15d0000, Classes={…}, IOMalloc allocation=0x8a7425d}

IOMasterPort(MACH_PORT_NULL, &master_port);
root_entry = IORegistryGetRootEntry(master_port);
IORegistryEntryCreateCFProperties(root_entry, &properties, kCFAllocatorDefault, kNilOptions);

// get IOKitDiagnostics
diagnostics = CFDictionaryGetValue(properties, CFSTR(kIOKitDiagnosticsKey));
classes = (CFDictionaryRef)CFDictionaryGetValue(diagnostics, CFSTR("Classes"));
CFDictionaryApplyFunction(classes, collect_service, NULL);
```

UserClients collection limits

- iOS
  - non-sandbox env e.g: jailbreak/SRD
- macOS
  - macOS sandboxed App
  - non-sandbox env e.g: terminal

Collection method comparison

```
➜  KidFuzzer git:(master) ✗ cd resources
➜  resources git:(master) ✗ cat service.xml | wc -l
     304
➜  resources git:(master) ✗ cat legacy/service1.xml | wc -l
     156
➜  resources git:(master) ✗
```

Quick review of KidFuzzerV1

- deep_fuzz
  - Input data Mutation
  - Race/Shm/… fuzz
  - scalarO/structO infoleak check
  - Driver independent backend by code audit

Differences in KidFuzzerV2

- deep_fuzz
  - Input data Mutation
  - Race/Shm/… fuzz
  - scalarO/structO infoleak check
  - ~~Driver independent backend by code audit~~
  - Backward fuzzing based components

Backward Fuzzing

- What does "Backward Fuzzing" mean?
  - Forward
  - Code audit -> find bug -> construct PoC
  - Backward
  - Public PoC -> extract pattern -> find bug

Backward Fuzzing

- What does "Backward Fuzzing" mean?
  - Forward
  - Code audit -> find bug -> construct PoC
  - Backward
  - Public PoC -> extract pattern -> find bug

Backward Fuzzing

- The Basic Idea
  - What happened in the past will happen in the future, what happens here happens there.
  - Extracts the code pattern or idea from the source
  - Reuse (time) or migrate (space) it to produce more bugs
- A bit of high level?
  - Let’s understand it by some real cases

Backward Fuzzing

- The Basic Idea
  - What happened in the past will happen in the future, what happens here happens there.
  - Extracts the code pattern or idea from the source
  - Reuse (time) or migrate (space) it to produce more bugs
- A bit of high level?
  - Let’s understand it by some real cases

Source of newly added components

- Code Diff and binary diff
- Public Proof-of-Concept
- Mind blown ideas through code audit

A 20 year old attack surface by code diff

```
@@ -571,7 +571,7 @@ IOReturn IOAudioControl::hardwareValueChanged(OSObject *newValue)
571  571              if (result == kIOReturnSuccess) {
572  572                  result = updateValue(newValue);
573  573              } else {
574      -                IOLog("IOAudioControl[%p]::hardwareValueChanged(%p) - Error 0x%x - invalid value.\n", this, newValue, result);
     574  +            IOLog("IOAudioControl::hardwareValueChanged - Error 0x%x - invalid value.\n", result);
575  575              }
576  576          }
577  577      } else {
@@ -628,12 +628,12 @@ IOReturn IOAudioControl::performValueChange(OSObject *newValue)
628  628              OSNumber *oldNumber, *newNumber;
629  629
630  630              if ((oldNumber = OSDynamicCast(OSNumber, getValue())) == NULL) {
631      -                IOLog("IOAudioControl[%p]::performValueChange(%p) - Error: can't call handler - int handler set and old value is not an OSNumber.\n",
     631  +            IOLog("IOAudioControl::performValueChange - Error: can't call handler - int handler set and old value is not an OSNumber.\n");
632  632                  break;
633  633              }
634  634
635  635              if ((newNumber = OSDynamicCast(OSNumber, newValue)) == NULL) {
636      -                IOLog("IOAudioControl[%p]::performValueChange(%p) - Error: can't call handler - int handler set and new value is not an OSNumber.\n",
     636  +            IOLog("IOAudioControl::performValueChange - Error: can't call handler - int handler set and new value is not an OSNumber.\n");
637  637                  break;
638  638              }
639  639
```

A 20 year old attack surface by code diff

- This bug introduced in IOAudioFamily-121.2.1 (macOS 10.1.5 2002)
- And it’s fixed in IOAudioFamily-205.11 (macOS 10.12 2016)

A 20 year old attack surface by code diff

- This bug introduced in IOAudioFamily-121.2.1 (macOS 10.1.5 2002)
- And it’s fixed in IOAudioFamily-205.11 (macOS 10.12 2016)
- But does the story end here?

A 20 year old attack surface by code diff

- This bug introduced in IOAudioFamily-121.2.1 (macOS 10.1.5 2002)
- And it’s fixed in IOAudioFamily-205.11 (macOS 10.12 2016)
- Or could be more?
  - The root cause would be like "Kernel developer might write the kernel address to log by accident"
  - It seems this could be a general attack surface in IOKit and even in XNU/Co-processors
  - Let’s examine this by KidFuzzerV2.0!

A 20 year old attack surface by code diff

- Variant bug hunting by KidFuzzerV2 (Unified Logging)
  - Trigger a lot of code paths (early_fuzz+deep_fuzz) ✅
  - An effective way to detect kernel info leak (log+grep) ✅

A 20 year old attack surface by code diff

- CVE-2022-42854 (IOBluetoothFamily)

```
__int64 __fastcall IOBluetoothHCIController::CreateSerialDevice(os_log_t *a1, unsigned __int8 *a2, __int64 a3){
__bzero(v20, 511LL);
 snprintf(v21, 0x1FFuLL, "**** [IOBluetoothFamily][CreateSerialDevice] -- calling device->start() -- device = %p ****\n", v15);
 _os_log_internal(&dword_0, a1[25], OS_LOG_TYPE_DEFAULT, "%s", v21);
}
```

```
kernel: (IOBluetoothFamily) [IOBluetoothFamily][CreateSerialDevice] -- calling device->start() -- device = 0xfffffe24cd78e600
kernel: (IOBluetoothFamily) [IOBluetoothFamily][CreateSerialDevice] -- calling device->init() -- device = 0xfffffe24cd78e800
```

A 20 year old attack surface by code diff

- CVE-2023-23500 (t600xdcp.im4p)

```
__cstring:0000000000574097 aAfkPSIfDIncmdD DCB "[AFK][%p:%s] if:%d inCmd:%d inResp:%d outRepErr:%d
outCmdErr:%d "
__cstring:0000000000574097                                      ; DATA XREF: __text:000000000002CE5C↑o
__cstring:0000000000574097                                      ; __text:000000000002CE9C↑o
__cstring:0000000000574097                              DCB "t:%lld",0
__cstring:00000000005740DE                      aUnknown        DCB "unknown",0         ; DATA XREF:
__text:000000000002CE4C↑o
```

```
[DCP:dpointInterface.cpp:606] [AFK][0xffffffff4100af08:AFKMailboxSharedMemoryEndpointInterface] if:133
inCmd:5 inResp:5 outRepErr:0 outCmdErr:0 t:13415177926
```

A 20 year old attack surface by code diff

- Silently patched (AppleFirmwareKit)

```
__int64 __fastcall AFKMailboxEndpointBase::setPowerState(AFKMailboxEndpointBase *this, __int64 a2, IOService *a3){
_os_log_internal(&dword_0, v8, OS_LOG_TYPE_DEBUG,
"%s(%s:%#llx): setPowerState:%lu _wake_msg:0x%llx device:%p epPowerState:%u assertionCount:%u\n",
ClassName, v11, RegistryEntryID, a2, *(_QWORD *)(*((_QWORD *)this + 23) + 80LL), a3,
*(unsigned int *)(*((_QWORD *)this + 23) + 72LL),
*(unsigned int *)(*((_QWORD *)this + 23) + 76LL));
}
```

DCPEndpoint(DCPEndpoint:0x10000062b): setPowerState:0 _wake_msg:0x0 device:0xfffffe1b336c6080 epPowerState:1 assertionCount:0 (Occasionally)

A 20 year old attack surface by code diff

- How about XNU?
  - In XNU, %p will be used only in some cases
  - Used in panic (no window to be used in exploit)
  - Used in printf but is commented
  - Not commented but used in XX_DEBUG which is not enabled for release versions

A 20 year old attack surface by code diff

- More than 20 bugs found by this attack surface, but Apple will merge the issue within a kext, so…
- CVE-2022-42854
- CVE-2023-23500
- CVE-2023-23501
- CVE-2023-23502
- CVE-2023-28184
- CVE-2023-32389
- More on the way…

Source of newly added components

- Code Diff and binary diff
- Public Proof-of-Concept
- Mind blown ideas through code audit

Extract high level root cause behind the bug

- E.g: limited resource management
  - MPTCP integer overflow
  - Manage limited resource is always a hard problem in kernel
  - We are gonna use this idea and migrate it to other parts in kernel space

Extract high level root cause behind the bug

- E.g: limited resource management
  - IOKit
  - Mach
  - BSD
  - ~~Co-processor~~

Extract high level root cause behind the bug

- E.g: limited resource management
  - IOKit
    - IOUserClient
  - Mach
  - BSD
  - ~~Co-processor~~

Extract high level root cause behind the bug

- User client null pointer

```
res = service->newUserClient( owningTask, (void *) owningTask,
            connect_type, propertiesDict, &client );

if (res == kIOReturnSuccess) {
if (!client->reserved) {
//…
```

Extract high level root cause behind the bug

- User client null pointer ☑️ watchdog timeout (xnu-8019.41.5)

```
if (res == kIOReturnSuccess && OSDynamicCast(IOUserClient, client)
== NULL) {
    res = kIOReturnError;
}
if (res == kIOReturnSuccess) {
if (!client->reserved) {
// e.g: AppleUpStreamUserClientDriver
```

Extract high level root cause behind the bug

- E.g: limited resource management
  - IOKit
    - IOUserClient
  - Mach
    - Mach ports
  - BSD
  - ~~Co-processor~~

Extract high level root cause behind the bug

- ipc_port integer overflow (latest version)
- IO_MAX_REFERENCES(0x7fffffff -> 0x0fffffff) iOS 15/macOS 12
- Send 0x2000 same port to 0x8000 different remote port
- 0x2000 * PORT_COUNT(0x8000) = 0x10000000 > 0x0fffffff

Extract high level root cause behind the bug

- ipc_port integer overflow (latest version)
- IO_MAX_REFERENCES(0x7fffffff -> 0x0fffffff) iOS 15/macOS 12
- Send 0x2000 same port to 0x8000 different remote port
- 0x2000 * PORT_COUNT(0x8000) = 0x10000000 > 0x0fffffff
- panic(cpu 4 caller 0xffffff01f80a440): os_refcnt: overflow (rc=0xffffffec180b81b4) @refcnt.c

Extract high level root cause behind the bug

- ipc_port integer overflow -> memory exhaustion (latest version)


```
  "build" : "iPhone OS 16.4.1 (20E252)",
  "product" : "iPhone13,2",
  "socId" : "8101",
  "socRevision" : "11",
  "incident" : "9D832D68-7211-4605-B9A9-2D88188FE4BA",
  "crashReporterKey" : "cc87293f54017db3e762d1c8098b0c0e68cd4383",
  "kernel" : "Darwin Kernel Version 22.4.0: Mon Mar  6 20:42:59 PST 2023; root:xnu-8796.102.5~1\/RELEASE_ARM64_T8101",
  "date" : "2023-04-26 15:52:46.43 +0800",
  "panicString" : "panic(cpu 0 caller 0xfffffff025b3bf54): kmem_alloc(0xffffffddafeb1d80, 49152, 0xe10141): failed with 3
@vm_kern.c:178\nDebugger message: panic\nMemory ID: 0x6\nOS release type: User\nOS version: 20E252\nKernel version:
Darwin Kernel Version 22.4.0: Mon Mar  6 20:42:59 PST 2023; root:xnu-8796.102.5~1\/RELEASE_ARM64_T8101\nFileset
Kernelcache UUID: 1BFAD8880EA0F9D92D88DF7ADA292863\nKernel UUID: 224DE1BA-AB41-38E3-8511-D1273096F638\nBoot session UUID:
9D832D68-7211-4605-B9A9-2D88188FE4BA\niBoot version: iBoot-8422.100.650\nsecure boot?: YES\nroots installed: 0\nPaniclog
version: 14\nKernelCache slide: 0x000000001d6b8000\nKernelCache base:  0xfffffff0246bc000\nKernel slide:
0x000000001d6c0000\nKernel text base:  0xfffffff0246c4000\nKernel text exec slide: 0x000000001e424000\nKernel text exec
```

Extract high level root cause behind the bug

- E.g: limited resource management
  - IOKit
    - IOUserClient
  - Mach
    - Mach ports
  - BSD
    - Dev drivers
  - ~~Co-processor~~

Extract high level root cause behind the bug

- /dev/perfmon OOB Access (latest version)

```
While(1) {open("/dev/perfmon_core", O_RDONLY);}
```

```
if (dmin >= perfmon_kind_max || dmin < 0) {
    panic("perfmon: invalid minor dev number: 0x%x", dev);
}
return &perfmon_devices[source_index][dmin];
```

Extract high level root cause behind the bug

- /dev/perfmon OOB Access -> mutex lock panic (latest version)


```
{"bug_type":"210","timestamp":"2023-04-18 14:35:06.00 +0800","os_version":"macOS 13.4
(22F5037d)","roots_installed":0,"incident_id":"C8D5594E-0E86-456C-B68C-C694CF2D53E8"}
{
  "build" : "macOS 13.4 (22F5037d)",
  "product" : "Mac13,1",
  "socId" : "6001",
  "socRevision" : "11",
  "incident" : "C8D5594E-0E86-456C-B68C-C694CF2D53E8",
  "crashReporterKey" : "583C5AAA-F9BA-FBBE-9308-CC4EE46A74B0",
  "kernel" : "Darwin Kernel Version 22.5.0: Sun Apr  2 19:22:28 PDT 2023; root:xnu-8796.120.31~10\/
RELEASE_ARM64_T6000",
  "date" : "2023-04-18 14:35:06.95 +0800",
  "panicString" : "panic(cpu 1 caller 0xfffffe0013c5bbe8): Mutex 0xfffffe00167da448 is unexpectedly owned by thread
0xfffffe29ea6c9030 @lock_mtx.c:175\nDebugger message: panic\nMemory ID: 0x6\nOS release type: User\nOS version:
22F5037d\nKernel version: Darwin Kernel Version 22.5.0: Sun Apr  2 19:22:28 PDT 2023; root:xnu-8796.120.31~10\/
RELEASE_ARM64_T6000\nFileset Kernelcache UUID: 9A2F6E3FE901C23580C1C9EA7094605D\nKernel UUID:
```

Source of newly added components

- Code Diff and binary diff
- Public Proof-of-Concept
- Mind blown ideas through code audit

Ret2leak

- CVE-2022-30916

IOReportUserClient::close:

```
// …
mov     rax, [rax+5D8h]
jmp     rax             <---- call IOService::close, but it does not has the return value
pop     rbp
retn                    <---- the rax is totally depending on IOService::close

// void IOService::close(IOService *   forClient, IOOptionBits  options)
```

Other tales about Co-processor bugs

- CVE-2023-28186 early_fuzz(AppleCLCD2) nullpointer/active panic

"panicString" : "panic(cpu 1 caller 0xfffffe00275d859c): DCP PANIC - IOMFB int_handler_gated: failure: axi_rd_err [0x40310634]\n - iomfb_driver(7)\nIOMFB int_handler_gated: failure: axi_rd_err [0x40310634]\n\nRTKit: RTKit-2062.40.5.debug - Client: local-t600xdcp.release\n!UUID: b9f40618-d5de-306f-bcd2-42f07dc6842b\nTime: 0x000000051ed17b64\n\nFaulting task 7 Call Stack: 0x0000000001e3ec 0x0000000001dde4 0x0000000001dbdc 0x00000000020ef0 0x0000000013d184 0x00000000000e0378 0x0000000013c6b4 0x0000000007d944 0x0000000001e880 0x00000000000116b0 0000000000000000\n

Other tales about Co-processor bugs

- CVE-2023-???? deep_fuzz(AppleXXX)

"panicString" : "panic(cpu 2 caller 0xfffffe00173e311c): XXX DATA ABORT @ 0x01054c70
pc=0x000000000112bdd8 Exception class=0x25 iss=0x4f far=0x000000000108bc60
far_physical=0x0000000293c8bc60\nRTKit: RTKit-2062.40.13.debug - Client: XXX\n!UUID:
55329378-843a-3ed2-bb6b-422efa54eb89\nTime: 0x000000270fa9fe7d\n\nFaulting task stack frame:\n
pc=0x000000000112bdd8 Exception class=0x25, iss=0x4f far=0x000000000108bc60
far_physical=0x0000000293c8bc60\n r00=0x00000000010d2f80 r01=0x000000000000002c
r02=0x000000000108bcd0 r03=0x0000000000000400\n r04=0xfffffffe51be000 r05=0x00000000010da748
r06=0x00000000010da558 r07=0x0000000001012400\n r08=0x000000000112bdd8
r09=0x000000000112c5b4 r10=0x000000000112c248 r11=0x0000000000000418\n
r12=0x0060000000000040 r13=0x0000000000008e42 r14=0x0000000000000001
r15=0000000000000000\n r16=0x0000000000000044 r17=0x0000000000000001
r18=0000000000000000 r19=0x00000000010da748\n r20=0x00000000010da760 r21=0xfffffffe51be000
r22=0x000000000108c150 r23=0x00000000010d2f80\n r24=0x0000000000000400 r25=0xfffffffe51be000
r26=0x00000000010da558 r27=0xfffffffe51be000\n r28=0x000000000000002c
r29=0x000000000108c130\n sp=0x000000000108bcc0 lr=0x000000000112c26c

Other tales about Co-processor bugs

- Co-processor bugs would be hard to understand from AP side
- Sometimes it just write a user controlled value to a MMIO region
- Or it takes a memory descriptor and start DMA

Other bugs found by KidFuzzerV2

- CVE-2022-42820 IOHIDFamily, OOB
- CVE-2022-42833 GPU Driver, MemDescriptor handling
- CVE-2023-28185 USB Driver, integer overflow
- CVE-2023-28187 SIO, driver and firmware

Summary

Summary

- Push one and pop more, variant bug hunting by fuzzing could be easy if you get the idea behind the bug
- Some bugs are hard to be found by code audit but might be easy by fuzzer
- Code audit and fuzzing are not opposite, but mutually reinforcing

References

- https://github.com/star-sg/Presentations/blob/main/POC%202022/Zhenpeng%20Pan.pdf
- https://github.com/potmdehex/slides/blob/main/Hexacon_2022_More_Tales_from_the_iOS_macOS_Kernel_Trenches.pdf
- https://bugs.chromium.org/p/project-zero/issues/detail?id=2004
- https://www.youtube.com/watch?v=8mQAYeozl5I
- https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf

Thank you !