(P)FACE into the Apple core and exploit to root

--Fuzzing with context enlightenment and exploit OSX IOKit vulnerabilities for fun and profit

Agenda

- Who we are
- Passive fuzzing framework
- Context enlightenment
- Exploit to root

Who are we

- Jack Tang:
- 10 years of anti-malware solution development
- Familiar with Windows/Mac kernel technology, browser and document exploit.
- Current focusing on research about Mac vulnerability and exploit

- Moony Li:
- 7 years of security production development
- RD Leader of Sandcastle core engine of DD(Deep Discovery) production for Gateway 0day exploit detection.
- Current focusing on research about Mac/Windows kernel vulnerability and exploit

And so what?

- Here below is the CVE and ZDI list until now(**NOT including submitted but pending**):
- CVE-2015-3787, CVE-2015-5867, CVE-2015-7021,CVE-2015-7020, CVE-2016-1716,ZDI-CAN-3536,ZDI-CAN-3558, ZDI-CAN-3598,ZDI-CAN-3596,ZDI-CAN-3603,CVE-2015-7067, CVE-2015-7076,CVE-2015-7106,CVE-2015-7109,CVE-2016-1718,CVE-2016-1747,CVE-2016-1749,CVE-2016-1753, ZDI-CAN-3693, ZDI-CAN-3694, CVE-2016-1795, CVE-2016-1808, CVE-2016-1810, CVE-2016-1817, CVE-2016-1820, CVE-2016-1798, CVE-2016-1799, CVE-2016-1812, CVE-2016-1814, CVE-2016-1818, CVE-2016-1816

Phase x: Fuzzing framework

Agenda

- Passive fuzzing framework
  - Previous work
  - Approach & consideration
  - Implementation
  - Best Practice

Previous work 1/2

- Traditional fuzzing by IOKit interface

Usually open the IOKit service name which they want to test, and pour fuzzing data into by the IOKit usermode API (e.g. IOConnectCallMethod)

  - Call sequence dependency
    - AppleCamIn (OpenDevice, PowerOnCamera…)
  - Input data dependency
    - AppleHDAEngineInput(input as user mode buffer pointer)
  - Timing dependency
    - IOHDIXHDDriveOutKernel(mount dmg)

Previous work 2/2

- Code review of target kernel extension

This costs much effort to reverse engineering binary code and in the face of so many IOKit services and userclient.

-Un-scalable

-Cost effort RE (upgrade)

Approach & Re-thinking

http://www.chinapoesy.com/gongxiangaebf830f-e011-4375-9312-af80aa2f184a.html

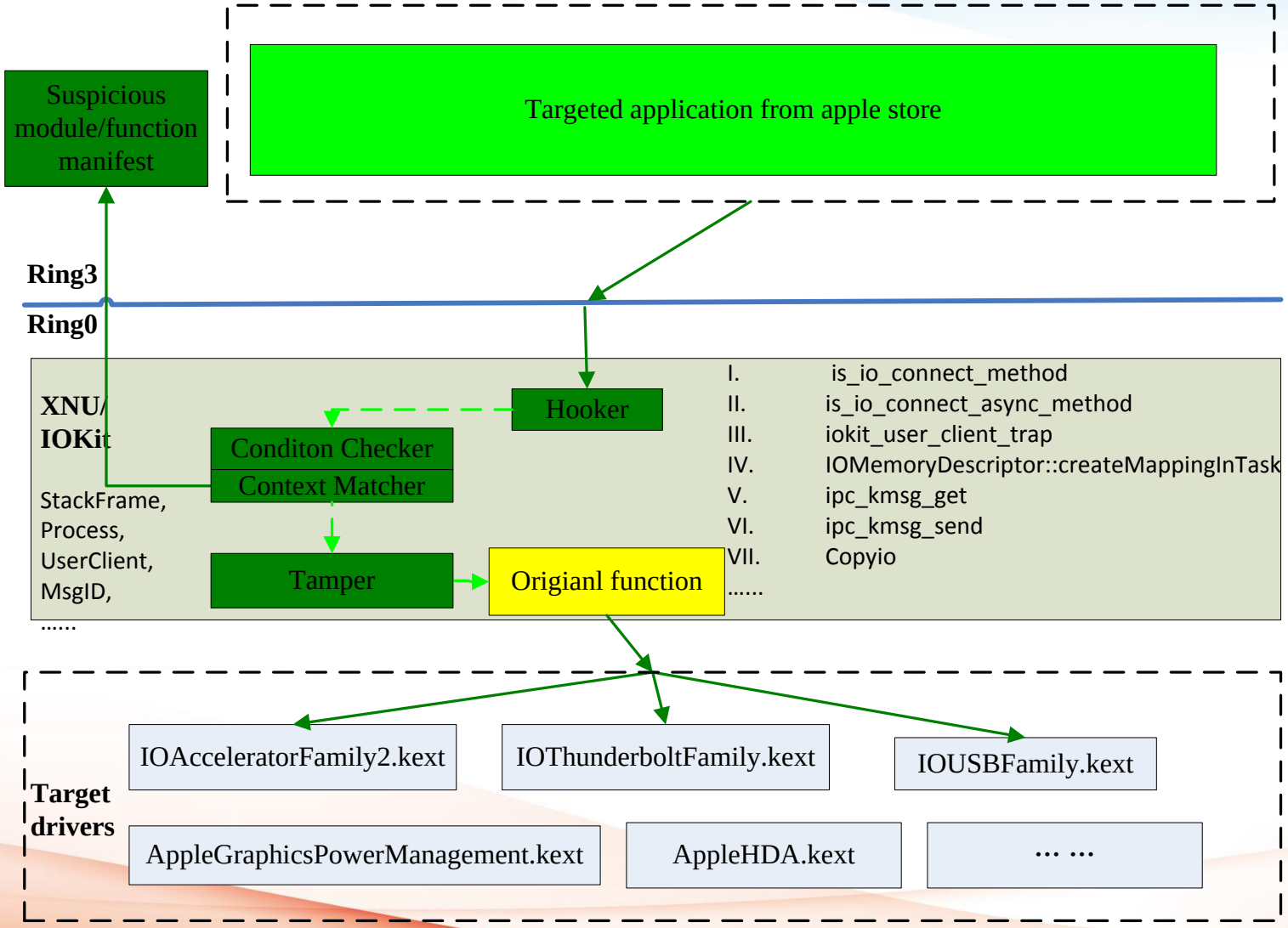
Approach & Re-thinking

- http://s1.sinaimg.cn/large/48f6d2f7gd04bcc333b70&690

Passive & Context

- “Passive”
  - means we don’t generate data to pour to interface from user mode. Whereas, we only trace at some key point in the kernel processing IOKit input data from user mode and tamper the data at proper time and location with restricted condition.
- “Context”
  - means a pattern in which scenario suspicious vulnerability is more likely to be existing.

Architecture overview



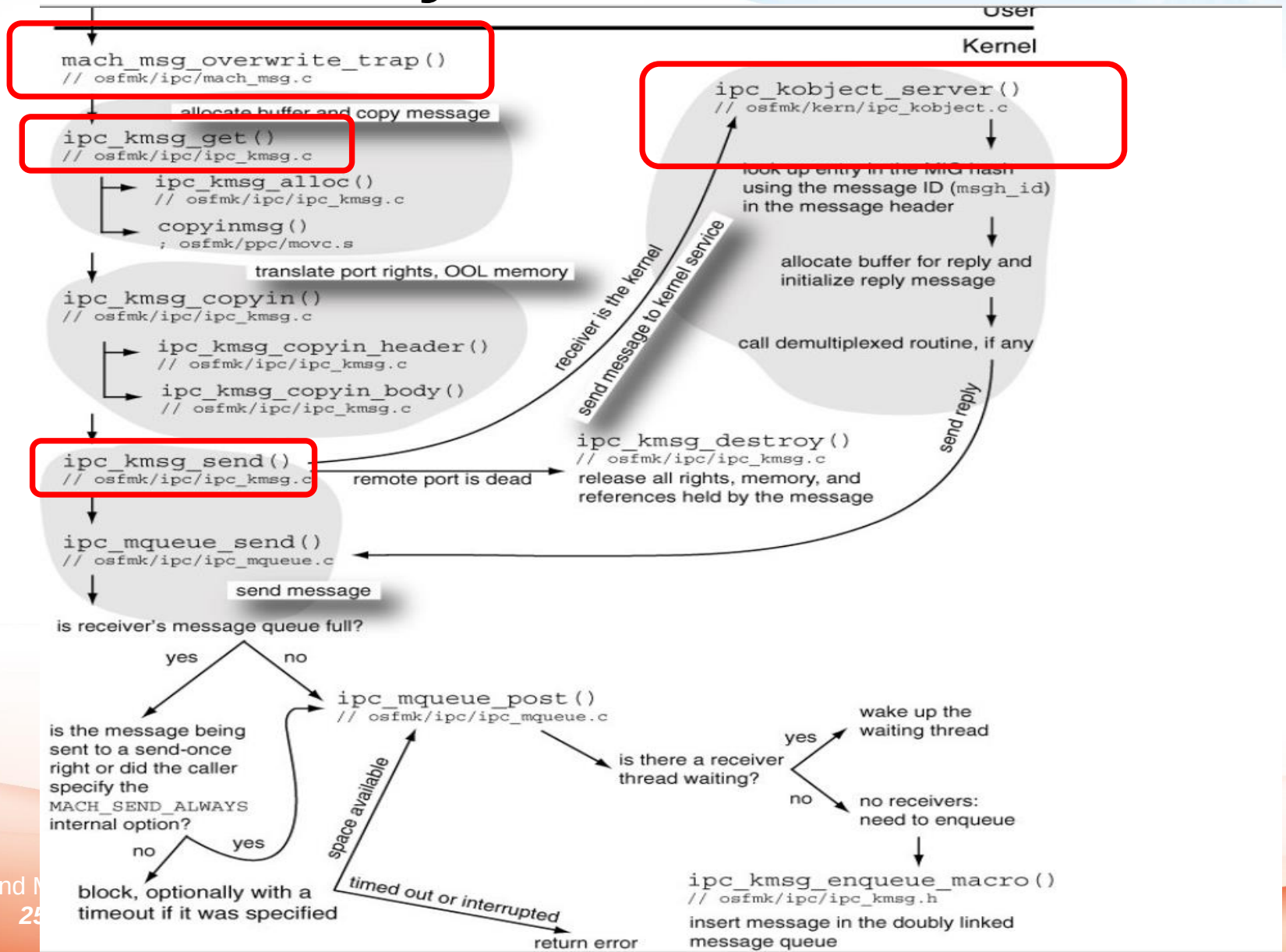
Pseudo

```
TargetAPI(params):
    //Call Original_TargetAPI(params)
    if (!ConditionChecker(params))  goto _exit();
    if (ContextMatcher(params))
            report alert;
    if (random())    tamper(params);
    Call Original_TargetAPI(params);
```

Hooker & Tamper

- The hooker
  - Direct taint-able from user
  - Hook one for many processes
    - Inline hook in kernel mode.
- Tamper
  - Only fuzzing the buffer content touchable by user
    - Inband_input, scalar_input, ool_input
    - NOT size

Layered Hook



Snappet

```
(lldb) showcurrentstacks

    Backtrace:
    kernel_stack = 0xffffff8876790000
    stacktop = 0xffffff8876793430
    0xffffff8876793430 0xffffff801679e37e Debugger((const char *) message = <>, )
    0xffffff88767934b0 0xffffff80166838c7 panic((const char *) str = <>, )
    0xffffff8876793690 0xffffff8016798fd9 panic_trap [inlined]((x86_saved_state64_t *) regs = <>, , (uint32_t) pl = <>, )
    0xffffff8876793690 0xffffff8016798daf kernel_trap((x86_saved_state_t *) state = <>, , (uintptr_t *) lo_spp = <>, )
    0xffffff88767936b0 0xffffff80167b7d83 kernel.development`trap_from_kernel + 0x26
    0xffffff88767937f0 0xffffff801650e05b kernel.development`memcpy + 0xb
    0xffffff88767937f0 0xffffff7f0000000f None + 0xffffff7f0000000f
    0xffffff8876793830 0xffffff7f96ed7d4d com.vmware.kext.VMwareGfx + 0x6d4d
    0xffffff88767938d0 0xffffff8016cb9657 ::shim_io_connect_method_scalarI_scalarO(IOExternalMethod *, IOService *, const io_user_scalar_t *,
mach_msg_type_number_t, io_user_scalar_t *, mach_msg_type_number_t *)((IOExternalMethod *) method = <>, , (IOService *) object = <>, , (const
io_user_scalar_t *) input = <>, , (mach_msg_type_number_t) inputCount = <>, , (io_user_scalar_t *) output = <>, , (mach_msg_type_number_t *) outputCount =
<>, )
    0xffffff8876793930 0xffffff8016cbbe28 IOUserClient::externalMethod(unsigned int, IOExternalMethodArguments*, IOExternalMethodDispatch*, OSObject*,
void*)((IOUserClient *) this = <>, , (uint32_t) selector = <>, , (IOExternalMethodArguments *) args = <>, , (IOExternalMethodDispatch *) dispatch = <>, ,
(OSObject *) target = <>, , (void *) reference = <>, )
    0xffffff8876793a70 0xffffff8016cb8f67 ::is_io_connect_method(io_connect_t, uint32_t, io_user_scalar_t *, mach_msg_type_number_t, char *,
mach_msg_type_number_t, mach_vm_address_t, mach_vm_size_t, char *, mach_msg_type_number_t *, io_user_scalar_t *, mach_msg_type_number_t *,
mach_vm_address_t, mach_vm_size_t *)((io_connect_t) connection = 0xffffff88767939b0, (uint32_t) selector = 10, (io_user_scalar_t *) scalar_input = <>, ,
(mach_msg_type_number_t) scalar_inputCnt = <>, , (char *) inband_input = <>, , (mach_msg_type_number_t) inband_inputCnt = 0, (mach_vm_address_t)
ool_input = <>, , (mach_vm_size_t) ool_input_size = <no location, value may have been optimized out>, , (char *) inband_output = <no location, value may have
been optimized out>, , (mach_msg_type_number_t *) inband_outputCnt = <no location, value may have been optimized out>, , (io_user_scalar_t *) scalar_output
= <>, , (mach_msg_type_number_t *) scalar_outputCnt = <no location, value may have been optimized out>, , (mach_vm_address_t) ool_output = <>, ,
(mach_vm_size_t *) ool_output_size = <>, )
    0xffffff8876793c40 0xffffff7f986a8f74 trampline_is_io_connect_method((io_connect_t) connection = 0xffffff801fbc0600, (uint32_t) selector = 10,
(io_user_scalar_t *) scalar_input = 0xffffff801eeb9910, (mach_msg_type_number_t) scalar_inputCnt = 2, (char *) inband_input = 0xffffff801eeb9924 "",
(mach_msg_type_number_t) inband_inputCnt = 0, (mach_vm_address_t) ool_input = 0, (mach_vm_size_t) ool_input_size = 0, (char *) inband_output =
0xffffff801d6ac600 "", (mach_msg_type_number_t *) inband_outputCnt = 0xffffff801d6ac5fc, (io_user_scalar_t *) scalar_output = 0xffffff8876793ca0,
(mach_msg_type_number_t *) scalar_outputCnt = 0xffffff8876793c9c, (mach_vm_address_t) ool_output = 0, (mach_vm_size_t *) ool_output_size =
0xffffff801eeb9944)
    0xffffff8876793d50 0xffffff801675c050 _Xio_connect_method((mach_msg_header_t *) InHeadP = <>, , (mach_msg_header_t *) OutHeadP =
0xffffff801d6ac5d0)
    0xffffff8876793d80 0xffffff8016687f73 ipc_kobject_server((ipc_kmsg_t) request = 0xffffff801eeb9880)
    0xffffff8876793dc0 0xffffff8016663ea3 ipc_kmsg_send((ipc_kmsg_t) kmsg = <>, , (mach_msg_option_t) option = <>, , (mach_msg_timeout_t) send_timeout =
0)
    0xffffff8876793e50 0xffffff7f9869f6dd trampline_ipc_kmsg_send((ipc_kmsg_t) kmsg = 0xffffff801eeb9880, (mach_msg_option_t) option = 3,
(mach_msg_timeout_t) send_timeout = 0)
    0xffffff8876793ec0 0xffffff801667a4c5 mach_msg_overwrite_trap((mach_msg_overwrite_trap_args *) args = <>, )
    0xffffff8876793f10 0xffffff7f986a345c trampline_mach_msg_overwrite_trap((mach_msg_overwrite_trap_args *) args = 0xffffff8876793f28)
    0xffffff8876793fb0 0xffffff80167828e0 mach_call_munger64((x86_saved_state_t *) state = 0xffffff801de4bca0)
    0x0000000000000000 0xffffff80167b85a6 kernel.development`hndl_mach_scall64 + 0x16
    stackbottom = 0xffffff8876793fb0
```

Hook Summary

- (Driver interface)is_io_connect_method
- (Driver interface)is_io_connect_async_method
- (Kernel)iokit_user_client_trap
- (Kernel)IOMemoryDescriptor::createMappingInTask
- (Mach Msg)ipc_kmsg_get
- (Mach Msg)ipc_kmsg_send
- (General IO)Copyio
- …

Why condition checker

1. Keep fuzzing stable
   - Get rid of noise (busy call, reproduced crashes)

2. Hunt vulnerability according to context

Dimension of condition 1/3

- &&, ||, *(wild match), white(black)

- Process
  - User id (root/Non-root)
  - Process Name (e.g. Safari, RCE, sandbox-evasion)
- Module
  - Module Name
- Function
  - Symbol Name/Address
  - Offset range

Dimension of condition 2/3

- Data
  - is_address_RWX
  - Copy direction(in/out)
  - Kernel or User space (SMAP noise)
- Call-Stack
  - Function ret
  - Stack Level (from bottom to top)
  - Level range[,]

Dimension of condition 3/3

- Misc
  - Mach_msg
    - msg subsystem id…
  - Userclient
    - serviceName,ClassName,selector…

Stack Frame Condition Sample

```
stack_match_item_t stack_matcher_for_copyio[]={
        //If any item in list match, then match
    //{routineName, cache}, routineAddress, offSetFrom, offsetTo, levelLow, levelHigh

{{"_shim_io_connect_method_scalarI_scalarO",STACK_ANY_INTEGER},STACK_ANY_INTEGER,0,
0xC120-0xB8B0, STACK_ALL_LEVEL_RANGE},

{{"_shim_io_connect_method_scalarI_structureO",STACK_ANY_INTEGER},STACK_ANY_INTEGER,
0, 0xDB94-0xD5C0, STACK_ALL_LEVEL_RANGE},

{{"_shim_io_connect_method_scalarI_structureI",STACK_ANY_INTEGER},STACK_ANY_INTEGER,0,
0xEA97-0xE490, STACK_ALL_LEVEL_RANGE},

{{"_shim_io_connect_method_structureI_structureO",STACK_ANY_INTEGER},STACK_ANY_INTEGE
R,0, 0xF588-0xF270, STACK_ALL_LEVEL_RANGE},

{{"_is_io_connect_method",STACK_ANY_INTEGER},STACK_ANY_INTEGER,0, 0xb2a9-
0xaf10,STACK_ALL_LEVEL_RANGE},
                }
```

UserClient Condition Sample

```
detail_control_entry_t g_white_listing_detail_control[] ={
    // procName,uid,driverBundleName, driverClassName, selFunctionNO
//{"*",PROCESS_UID_ANY_INTEGER,"*","AGPMClient",7312},,
    {"*",PROCESS_UID_ANY_INTEGER,"*","IGAccelSharedUserClient",1},//crash-24
    {"*",PROCESS_UID_ANY_INTEGER,"*","AccelSurface",16},//crash-23
    {"*",PROCESS_UID_ANY_INTEGER,"*",OBJECT_CLASS_NAME_NO_FOUND,16},
    {"*",PROCESS_UID_ANY_INTEGER,"*","HD",2},//crash-21
    {"*",PROCESS_UID_ANY_INTEGER,"*","IX",2},//crash-21
    "*",PROCESS_UID_ANY_INTEGER,"*","AGPM",7312,//crash-11
    "*",PROCESS_UID_ANY_INTEGER,"*","IGAccelGLContext",2,//crash-28
```

Mach-msg Condition Sample

#define KMSG_IOKIT_SUBSYSTEM_RANGE 0xAF0, 0x0B47

detail_control_entry_for_ipc_kmsg_send_t g_black_listing_detail_control_foripc_kmsg_send[] ={

//procName,uid,msg_id_from, msg_id_to, routineName, addr, addr_offset_from, addr_offset_to

"chrome",PROCESS_UID_ANY_INTEGER,
KMSG_IOKIT_SUBSYSTEM_RANGE,"__Xio_connect_method",KMSG_ADDR_OFFSET_ANY_RANGE,KM
SG_LEAVING,};

- #define KMSG_IOKIT_SUBSYSTEM_RANGE 0xAF0, 0x0B47
- #define KMSG_MACH_VM_SUBSYSTEM_RANGE 0x12C0, 0x12D4
- #define KMSG_MACH_PORT_SUBSYSTEM_RANGE 0xC80, 0x0CA4
- #define KMSG_MACH_HOST_SUBSYSTEM_RANGE 0xC8, 0xE4
- #define KMSG_HOST_PRIV_SUBSYSTEM_RANGE 0x190, 0x1AA
- ……

What's Context

- Pattern accumulated in bug hunting experience
- Shedding more enlighten for code review
  - Buggy module, interface for RE.
- Not vulnerability but indicating suspicious vulnerability
- Implemented through condition checker

Context Sample

- Some IOKit related memory corruption vulnerabilities would happen in the following context:
  - Call IOMemoryDescriptor :: createMappingInTask to mapping user mode buffer space to kernel mode.
  - Read a value from the buffer and use it as a size to read or write a buffer.
- Some kernel information leak vulnerability would happen in the following context:
  - The output buffer's content has 0xFFFFFF prefix.

Best Practice 1/3

- Fuzzing Source:
  - Multiple application
    - AppStore (MMORPG games, FaceTime,USB hardisk, BlueTooth, Wifi, VM,DirectX…)
    - Virus Total, Apple OpenSource UT, github sample code
  - Combination of rich kind of fuzzing source
    - Active fuzzing, Python watchdog, browsing WebGL
- Fuzzing Stability:
  - Bypass active hang, black screen, reproduced cases using condition checker(nvTestlaSurfaceTesla, IGAccelGLContext, IGAccelSurface…)

Best Practice 2/3

- Reproduction:
  - Log through network
  - Log to NVRAM? Log to memory and kdp_panic_dump callback?
  - Core dump server
    - sh-3.2# nvram boot-args=”pmuflags=1 debug=0xd44 kext-dev-mode=1 kcsuffix=development –v _panicd_ip=10.64.80.106”
  - Thunderbolt+fwkdp+lldb
- Automation
  - kdp_panic_dump callback+dump+reboot
  - VM(Vmware fusion…) revert

Best Practice 3/3

- Misc:
  - Keep fuzzing not SO busy(random maybe)
  - Hot run and fuzz on demand
  - Keep OS update with KDK


Phase x: Exploit to root

Obstacle

- ~~**SIP (System Integrity Protection)**~~
- **KALSR (Kean Team** method**)**
- **SMAP**
- **SMEP**

https://speakerdeck.com/marcograss/dont-trust-your-eye-apple-graphics-is-compromised

- A trick to do OSX kernel Heap Feng Shui
- Exploit to root with founded bugs

OSUnserializeXML

- The OSX/iOS hacking guru Stefan Esser (@i0n1c) propose *OSUnserializeXML* is a good way in SyScan 2012

```
<plist version="1.0">
<dict>
    <key>ThisIsOurData</key>
    <array>
        <data>VGhpcyBJcyBPdXIgRGF0YSB3aXRoIGEgTlVMPgA8+ADw=</data>
        <data format="hex">00112233445566778899aabbccddeeff</data>
        <data>...</data>
    </array>
</dict>
</plist>
```

https://reverse.put.as/wp-content/uploads/2011/06/SyScan2012_StefanEsser_iOS_Kernel_Heap_Armageddon.pdf

OSUnserializeXML cont.

- But in the most case, the *OSDictionary* is allocated by OSUnserializeXML will be freed by *OSObject::release* in one system call

```
lea     rdx, [rbp+var_30] ; OSString **
mov     rdi, rbx          ; char *
mov     rsi, r14          ; unsigned __int64
call    __Z16OSUnserializeXMLPKcmPP8OSString ; OSUnserializeXML(char const*,ulong,OSString **)
mov     r14, rax
mov     r15, r14
```

° ° °

```
mov     rdi, r15
call    qword ptr [rax+28h] ; OSObject::release
```

OSUnserializeXML cont.

- If the allocated object is referenced by other component, it will not be released even if call *object::release* on it.
- *IORegistry* is good choice.
- So we find *OSUnserializeXML* invoking nearby *IORegistry* method calling …

OSUnserializeXML cont.

- In IOKIT service *IOMProotDomain* , slector 7 (*kPMSleepSystemOptions*)

RootDomainUserClient::secureSleepSystemOptions

```
unserializedOptions = OSDynamicCast( OSDictionary,
                                     OSUnserializeXML((const char *)inOptions, inOptionsSize,
```

。。。

```
// Publish Sleep Options in registry under root_domain
fOwner->setProperty( kRootDomainSleepOptionsKey, unserializedOptions);
```

```
*returnCode = fOwner->sleepSystemOptions( unserializedOptions );

unserializedOptions->release();
```

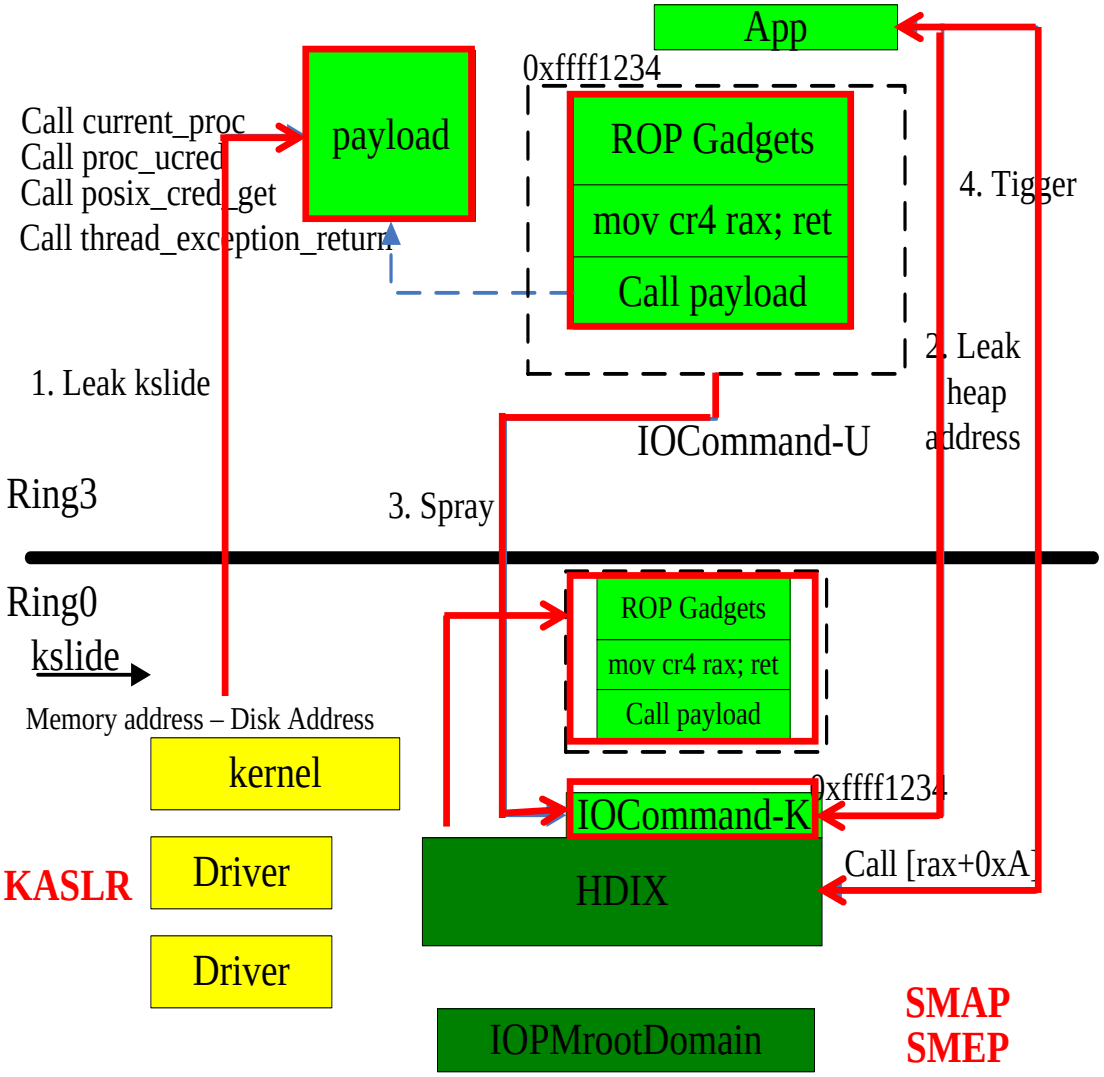

Exploit to root by founded
bugs and this trick

Bugs

- CVE-2016-1820 : In disk image module, it will take an object address and use a QWORD value in the object as function pointer to call.

- CVE-2016-xxxx: In disk image module, it will leak a object address, which exists in kernel heap.

Exploit Process



1. Use KEEN team's method to calculate KSLIDE.

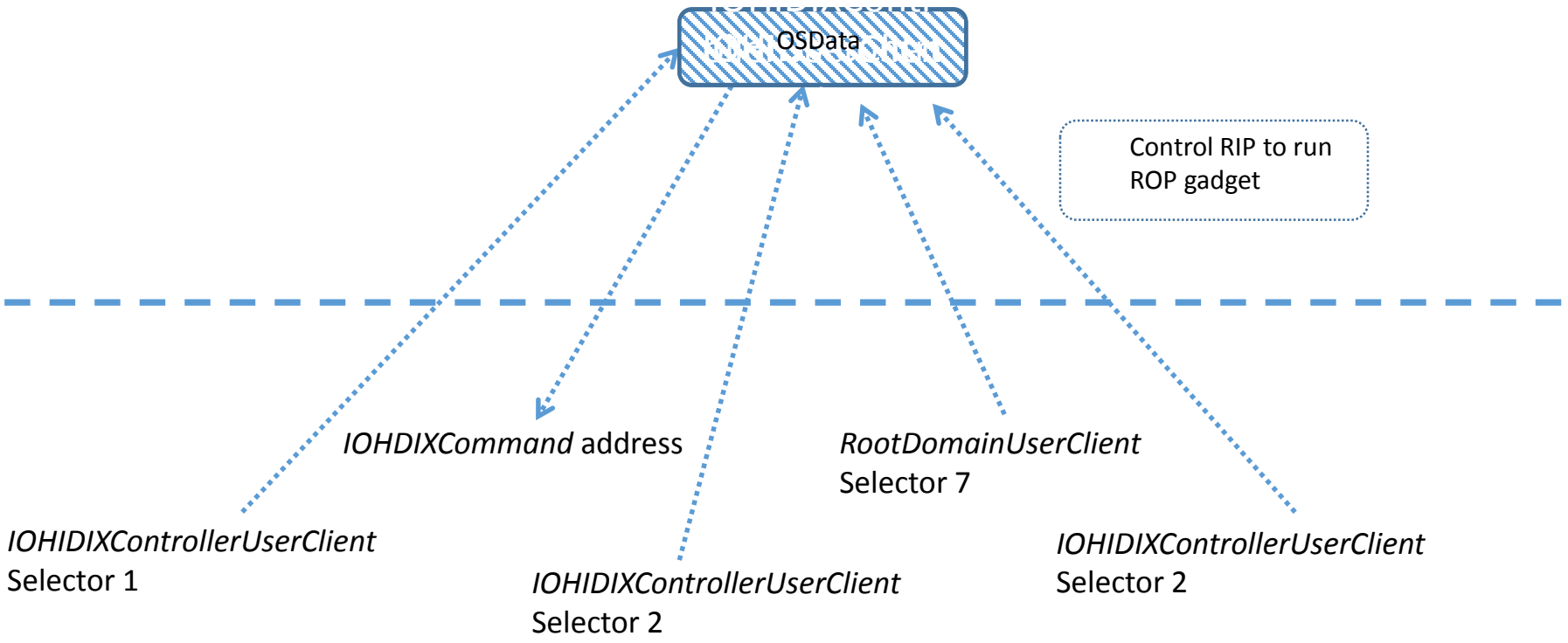
2. Call user client IOHIDIXControllerUserClient 's selector 1. From the output , we can get a kernel heap address of object IOHDIXCommand. Then call IOHIDIXControllerUserClient 's selector 2. It will release the object.

3. Call RootDomainUserClient user client 's selector 7 with a carefully prepared XML as parameter , which include ROP gadget in <data> part.

4. Call IOHIDIXControllerUserClient Selector 2 to get RIP execution

Exploit Process Detail

Kernel mode



User mode

Demo

Thanks very much