A Story of Exploiting macOS Sierra Kernel

Zhenquan Xu, Tielei Wang, Hao Xu of TEAM Pangu

Agenda

- New Mitigations in macOS Sierra
- Kernel Info Leak
- Kernel Code Execution
- Conclusion

macOS Sierra

- macOS Sierra (version 10.12) is the thirteenth major release of macOS

- released to end users on September 20, 2016

PwnFest 2016

- Nov 10th ~ 11th, 2016, Seoul, Korea

WHAT IS PWNFEST

PwnFest is a bug pwning 'festival' for better security organized by POC with the help of sponsors, vendors, and judges in 2016. You can enjoy PwnFest every year.

PwnFest 2016

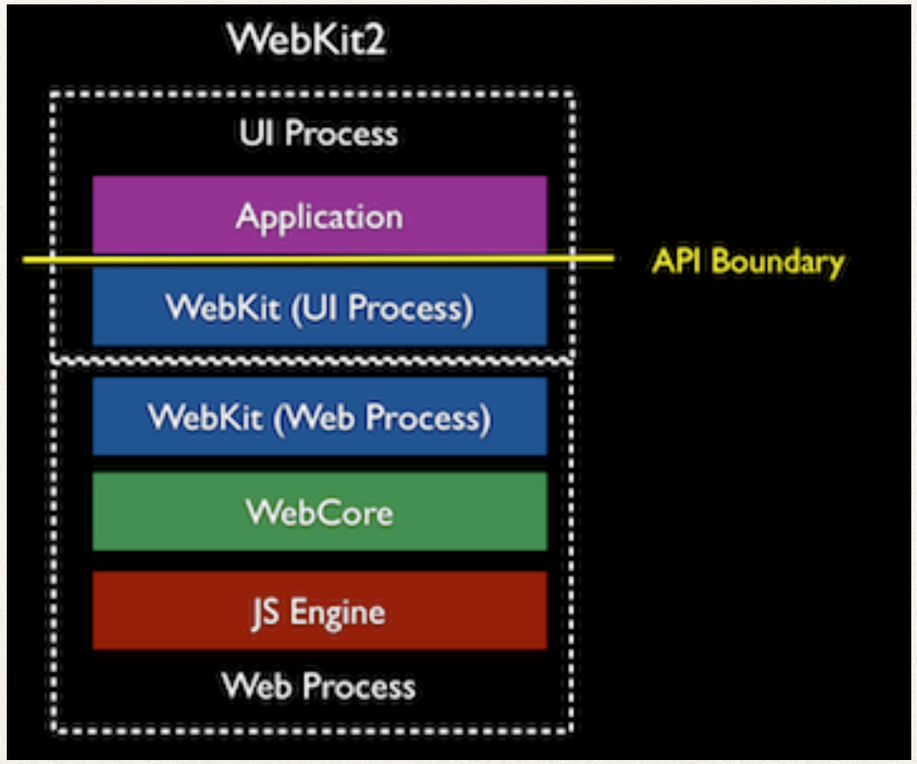
- Team up with Lokihardt for pwning Safari on macOS Sierra
  - Remote code execution in Safari by Lokihardt
  - Kernel code execution in Safari by us

Fixed in macOS Sierra 10.12.3

- Released January 23, 2017
- CVE-2017-2358
- CVE-2017-2357

Overview of Pwn Process

- Safari process model is based on WebKit2, splitting into different families

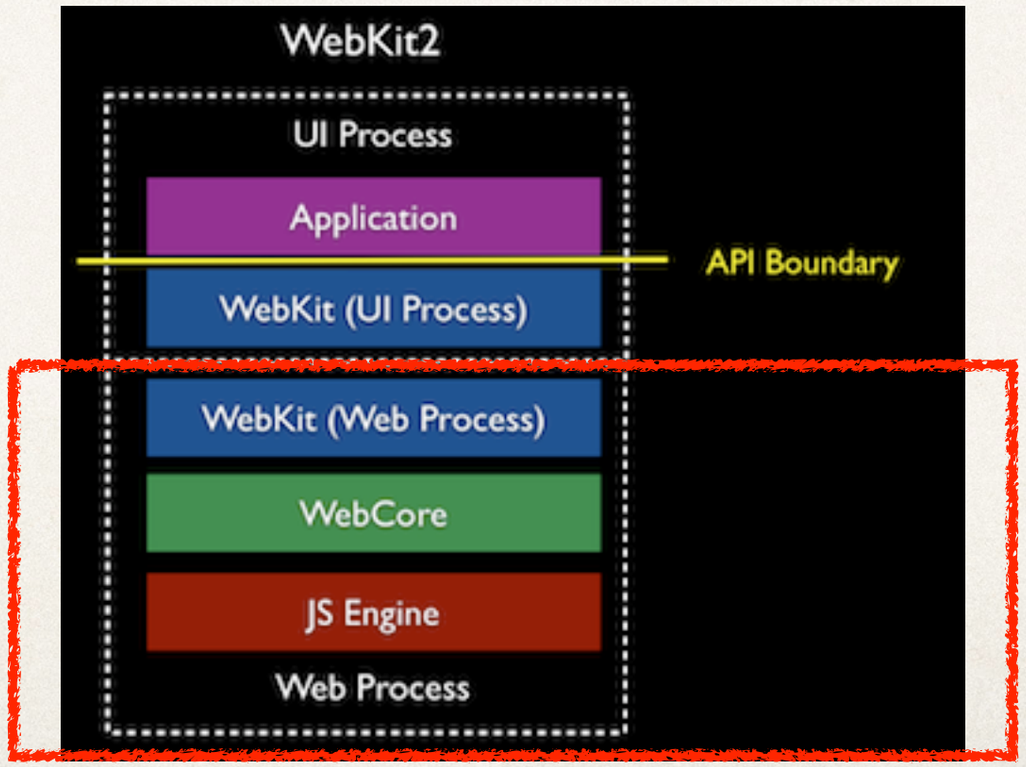


https://googleprojectzero.blogspot.co.id/2014/11/pwn4fun-spring-2014-safari-part-ii.html

https://trac.webkit.org/wiki/WebKit2

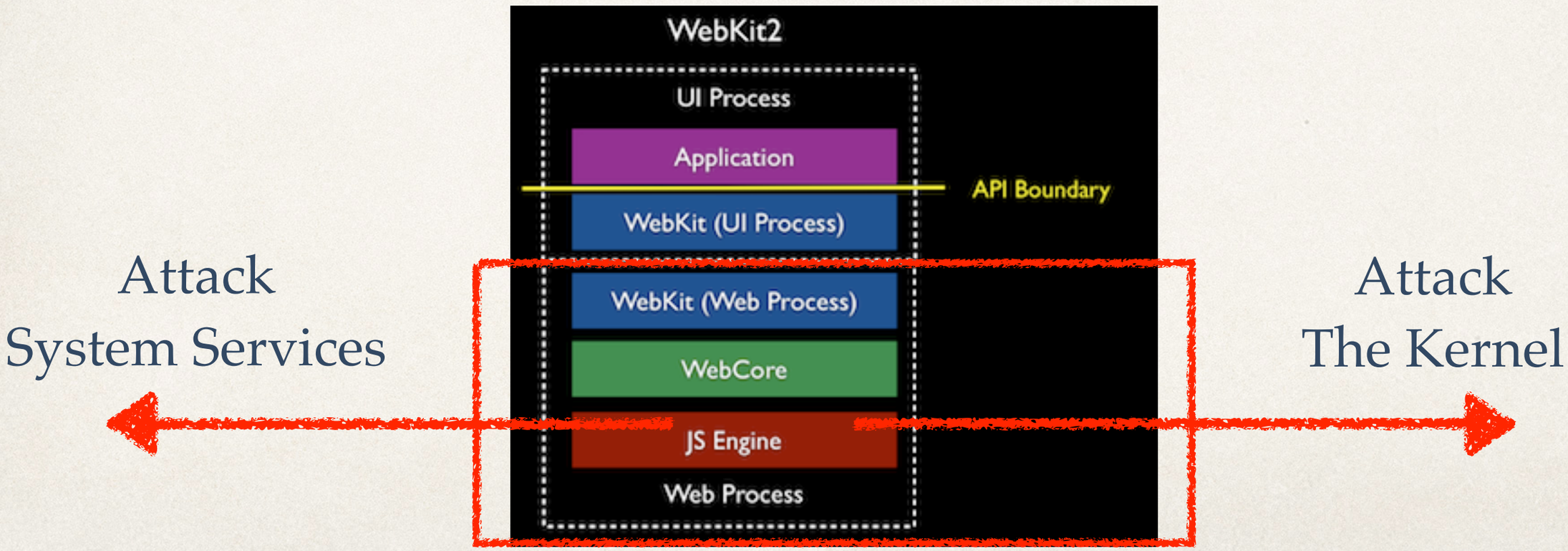
Overview of Pwn Process

- WebProcess is responsible for loading, parsing, rendering, thus is the main target
- But WebProcess is confined by sandbox



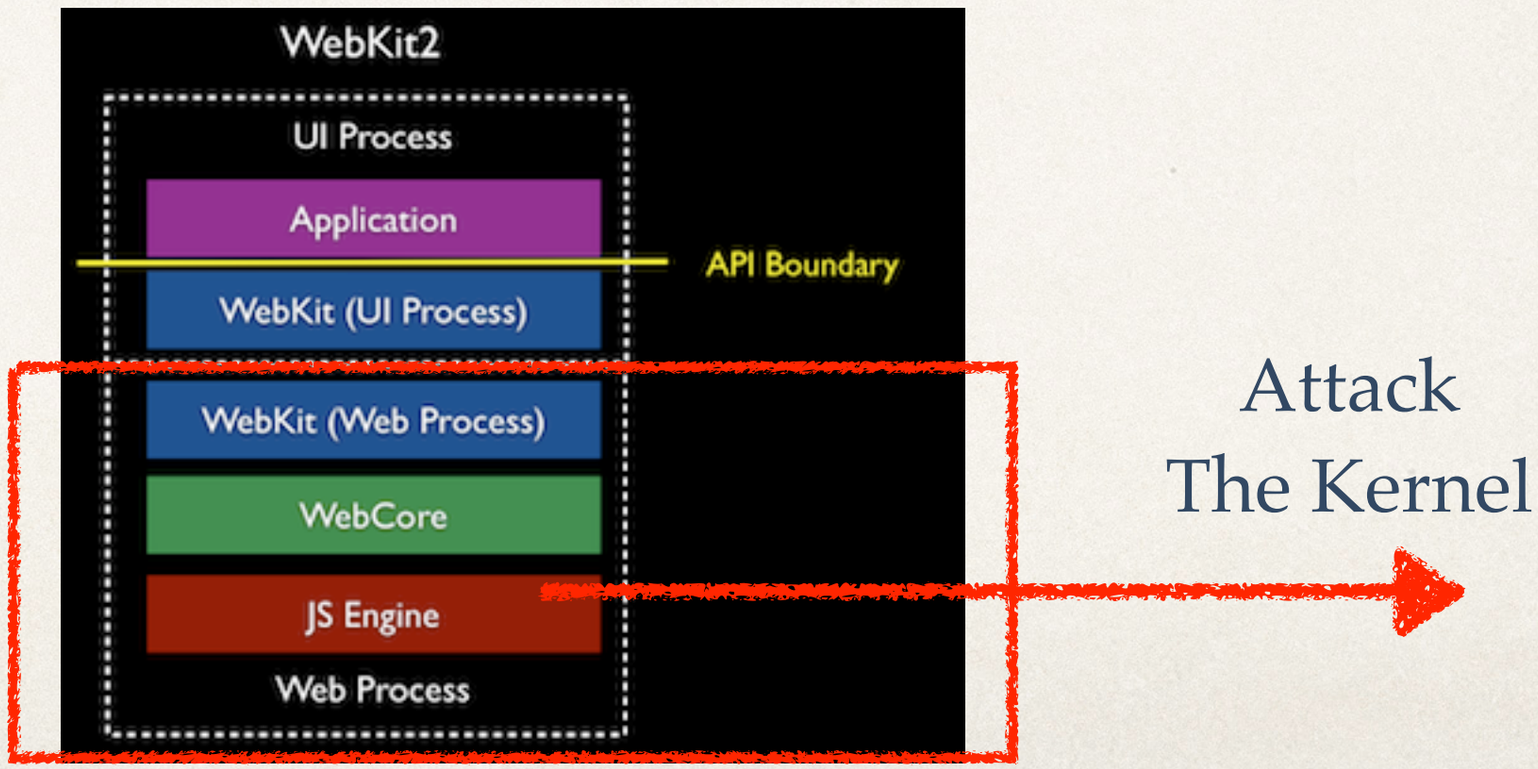
Overview of Pwn Process

- After gaining RCE in WebProcess, we need to either exploit other system services via IPC (e.g., WindowServer, fontd), or exploit kernel vulnerabilities directly



Overview of Pwn Process

- We decide to directly exploit kernel vulnerabilities



Review Kernel Attack Surfaces

- /System/Library/Frameworks/WebKit.framework/Versions/A/Resources/com.apple.WebProcess.sb
- /System/Library/Sandbox/Profiles/system.sb

```
;; IOKit user clients
(allow iokit-open
       (iokit-user-client-class "AppleUpstreamUserClient")
       (iokit-user-client-class "IOHIDParamUserClient")
       (iokit-user-client-class "RootDomainUserClient")
       (iokit-user-client-class "IOAudioControlUserClient")
       (iokit-user-client-class "IOAudioEngineUserClient"))
```

```
(allow iokit-open
       (iokit-connection "IOAccelerator")
       (iokit-registry-entry-class "IOAccelerationUserClient")
       (iokit-registry-entry-class "IOSurfaceRootUserClient")
       (iokit-registry-entry-class "IOSurfaceSendRight"))
;; CoreVideo CVCGDisplayLink
(allow iokit-open
       (iokit-registry-entry-class "IOFramebufferSharedUserClient"))
;; H.264 Acceleration
(allow iokit-open
       (iokit-registry-entry-class "AppleSNBFBUserClient"))
;; QuartzCore
(allow iokit-open
       (iokit-registry-entry-class "AGPMClient")
       (iokit-registry-entry-class "AppleGraphicsControlClient")
       (iokit-registry-entry-class "AppleGraphicsPolicyClient"))
```

Agenda

- **New Mitigations in macOS Sierra**
- Kernel Info Leak
- Kernel Code Execution
- Conclusion

New Mitigations

- No more io_service_open_extended heap spray

```
do
{
if (properties)
{
    OSObject *      obj;
    vm_offset_t     data;
    vm_map_offset_t map_data;

    if( propertiesCnt > sizeof(io_struct_inband_t))
    return( kIOReturnMessageTooLarge);

    err = vm_map_copyout( kernel_map, &map_data, (vm_map_copy_t) properties );
    res = err;
    data = CAST_DOWN(vm_offset_t, map_data);
    if (KERN_SUCCESS == err)
    {
    // must return success after vm_map_copyout() succeeds
    obj = OSUnserializeXML( (const char *) data, propertiesCnt );
    vm_deallocate( kernel_map, data, propertiesCnt );
    propertiesDict = OSDynamicCast(OSDictionary, obj);
    if (!propertiesDict)
    {
        res = kIOReturnBadArgument;
        if (obj)
        obj->release();
    }
    }
    if (kIOReturnSuccess != res)
    break;
}
```

```
    if (properties) return (kIOReturnUnsupported);
#if 0
    {
        OSObject *      obj;
        vm_offset_t     data;
        vm_map_offset_t map_data;

        if( propertiesCnt > sizeof(io_struct_inband_t))
        return( kIOReturnMessageTooLarge);

        err = vm_map_copyout( kernel_map, &map_data, (vm_map_copy_t) properties );
        res = err;
        data = CAST_DOWN(vm_offset_t, map_data);
        if (KERN_SUCCESS == err)
        {
        // must return success after vm_map_copyout() succeeds
        obj = OSUnserializeXML( (const char *) data, propertiesCnt );
        vm_deallocate( kernel_map, data, propertiesCnt );
        propertiesDict = OSDynamicCast(OSDictionary, obj);
        if (!propertiesDict)
        {
            res = kIOReturnBadArgument;
            if (obj)
            obj->release();
        }
        }
        if (kIOReturnSuccess != res)
        break;
    }
#endif
```

New Mitigations

- Limited count of IOUserClient objects
  - After reaching the threshold, IOServiceOpen() reuturns an error

```
2017-04-07 14:11:14.446 test[51571:2757644] userclient count: 885, kr = 0x00000000 userclient port: 0x38b03
2017-04-07 14:11:14.446 test[51571:2757644] userclient count: 886, kr = 0x00000000 userclient port: 0x38c03
2017-04-07 14:11:14.446 test[51571:2757644] userclient count: 887, kr = 0x00000000 userclient port: 0x38d03
2017-04-07 14:11:14.447 test[51571:2757644] userclient count: 888, kr = 0x00000000 userclient port: 0x38e03
2017-04-07 14:11:14.447 test[51571:2757644] userclient count: 889, kr = 0x00000000 userclient port: 0x38f03
2017-04-07 14:11:29.447 test[51571:2757644] userclient count: 890, kr = 0xe00002c7 userclient port: 0x0
2017-04-07 14:11:44.447 test[51571:2757644] userclient count: 891, kr = 0xe00002c7 userclient port: 0x0
2017-04-07 14:11:59.447 test[51571:2757644] userclient count: 892, kr = 0xe00002c7 userclient port: 0x0
2017-04-07 14:12:14.448 test[51571:2757644] userclient count: 893, kr = 0xe00002c7 userclient port: 0x0
```

New Mitigations

- Changes in heap memory management
  - All zones use metadata now
    - Four queues of metadata:
      - any_free_foreign : pages from different zones
      - all_free : pages whose elements are all free
      - intermediate : pages contain both free and used elements
      - all_used : pages whose elements are all in use

New Mitigations

- Changes in heap memory management
  - Always poison tiny zones, periodically poison larger zones

```
/*
 * Poison the memory before it ends up on the freelist to catch
 * use-after-free and use of uninitialized memory
 *
 * Always poison tiny zones' elements (limit is 0 if -no-zp is set)
 * Also poison larger elements periodically
 */
vm_offset_t     inner_size = zone->elem_size;
uint32_t sample_factor = zp_factor + (((uint32_t)inner_size) >> zp_scale);
if (inner_size <= zp_tiny_zone_limit)
    poison = TRUE;
else if (zp_factor != 0 && sample_counter(&zone->zp_count, sample_factor) == TRUE)
    poison = TRUE;
```

```
zp_tiny_zone_limit = (vm_size_t) cpu_info.cache_line_size;
```

New Mitigations

- Changes in heap memory management
  - Pointer pointing to next free element is xor-ed with the zp_nopoison_cookie
- No more point-to-free-vtable attack

```
/*
 * Insert this element at the head of the free list. We also xor the
 * primary pointer with the zp_nopoison_cookie to make sure a free
 * element does not provide the location of the next free element directly.
 */
*primary                = old_head ^ zp_nopoison_cookie;
```

Agenda

- New Mitigations in macOS Sierra
- **Kernel Info Leak**
- Kernel Code Execution
- Conclusion

Quick View of IOKit

- The IOKit is a collection of system frameworks, libraries, tools, and other resources for creating device drivers

- Extensive user mode API (via IOKit.framework)
  - Direct memory mapping to user mode
  - UserClient/External method calls
  - Notification and messaging

Quick View of IOKit

- IOKitLib implements non-kernel task access to common IOKit object types - IORegistryEntry, IOService, IOIterator, etc
  - IOServiceGetMatchingService
    - Look up a registered IOService object that matches a matching dictionary
  - IOServiceOpen
    - Create a connection to an IOService, return an IOUserClient port
  - IOConnectSetNotificationPort
    - Set a port to receive notifications from an IOUserClient object
  - IOConnectCallMethod
    - Pass/get data to an IOUserClient object

Info Leak - Overview

- The vulnerability lies in IOAudioFamily (before 204.4)
- Source code:
  https://opensource.apple.com/source/IOAudioFamily/IOAudioFamily-204.4/
- The vulnerability is an uninitialized heap issue

Vulnerability Analysis

- IOAudioControlUserClient allows userspace programs to register a notification port and send a notification message to them when certain audio events happen

- IOConnectSetNotificationPort in userspace will reach IOAudioControlUserClient::registerNotificationPort in the kernel

Vulnerability Analysis

```
IOReturn IOAudioControlUserClient::registerNotificationPort(mach_port_t port, UInt32 type, UInt32 refCon)
{
    ...
    if (notificationMessage == 0) {
        notificationMessage = (IOAudioNotificationMessage *)
         IOMallocAligned(sizeof(IOAudioNotificationMessage), sizeof (IOAudioNotificationMessage *));
        if (!notificationMessage) {
            return kIOReturnNoMemory;
        }
    notificationMessage->messageHeader.msgh_bits = MACH_MSGH_BITS(MACH_MSG_TYPE_COPY_SEND, 0);
    notificationMessage->messageHeader.msgh_size = sizeof(IOAudioNotificationMessage);
    notificationMessage->messageHeader.msgh_remote_port = port;
    notificationMessage->messageHeader.msgh_local_port = MACH_PORT_NULL;
    notificationMessage->messageHeader.msgh_reserved = 0;
    notificationMessage->messageHeader.msgh_id = 0;
    notificationMessage->ref = refCon;
    ...
}

typedef struct _IOAudioNotificationMessage
{
    mach_msg_header_t  messageHeader;
    UInt32      type;
    UInt32      ref;
    void *      sender;
} IOAudioNotificationMessage;
```

notificationMessage is not zeroed out and the *type* and the *sender* field is not initialized

Vulnerability Analysis

- IOAudioControlUserClient allocates a notification message struct and sends it to userpspace programs via IOAudioControlUserClient::sendChangeNotification

Info Leak - Leak to Userland

```
void IOAudioControlUserClient::sendChangeNotification(UInt32 notificationType)
{
    if (notificationMessage) {
        kern_return_t kr;

        notificationMessage->type = notificationType;
        kr = mach_msg_send_from_kernel(&notificationMessage->messageHeader,
notificationMessage->messageHeader.msgh_size);
        if ((kr != MACH_MSG_SUCCESS) && (kr != MACH_SEND_TIMED_OUT)) {
            IOLog("IOAudioControlUserClient: sendRangeChangeNotification() failed - msg_send
returned: %d\n", kr);
        }
    }
}
```

- The notificationMessage is allocated in kalloc.72 zone
- The *sender* field offset is 0x38
- We can read 8-byte data at the offset of 0x38 from any object in kalloc.72

Exploiting The Vulnerability

- Create many IOAudioControlUserClient objects through IOServiceOpen
- Register notification ports on such IOAudioControlUserClient objects via IOConnectSetNotificationPort
  - each IOAudioControlUserClient object will allocate a notificationMessage struct with uninitialized 8 bytes data
- Trigger a notification event, and receive the notification messages

However…

- Challenge 1: How to trigger the notification event
- Challenge 2: How to leak critical information such as KALSR slide

Info Leak - Sandbox Restriction

- Setting sound volume level can trigger the notification
  - i.e., set “IOAudioControlValue” via IORegistryEntrySetCFProperties
- However, the webcontent sandbox profile does not allow to set this property

coreaudiod

- Who is allowed to set sound volume level?

- By greping IOAudioControlValue, we found /usr/sbin/coreaudiod can set sound volume level

- coreaudiod is also responsible for Mach Service com.apple.audio.coreaudiod

coreaudiod

- Webprocess can also talk with com.apple.audio.coreaudiod service
- As a result, we can instruct coreaudiod to trigger the notification event

```
;; Various services required by AppKit and other frameworks
(allow mach-lookup
       (global-name "com.apple.DiskArbitration.diskarbitrationd")
       (global-name "com.apple.FileCoordination")
       (global-name "com.apple.FontObjectsServer")



       (global-name "com.apple.PowerManagement.control")
       (global-name "com.apple.SystemConfiguration.configd")
       (global-name "com.apple.SystemConfiguration.PPPController")
       (global-name "com.apple.audio.SystemSoundServer-OSX")
       (global-name "com.apple.audio.VDCAssistant")
       (global-name "com.apple.audio.audiohald")
       (global-name "com.apple.audio.coreaudiod")
```

How to leak KALSR slide

- We can read 8-byte data at the offset of 0x38 from a freed object in kalloc.72

- How to make the 8-byte data meaningful?

Info Leak - Leak OSSerialize Object

```
class OSSerialize : public OSObject
{
    ...
private:
    char           * data;
    unsigned int     length;
    unsigned int     capacity;
    unsigned int     capacityIncrement;
    OSArray * tags;
    bool     binary;
    bool     endCollection;
    Editor editor;
    void * editRef;
    ...
}
```

- The OSSerialize objects locates in the same zone as notificationMessage object
- The member at offset 0x38 is a function pointer
- Use separate thread to spray OSSerialize objects and leak several times to ensure stability

```
typedef const OSMetaClassBase * (*Editor)(
                        void* reference,
                        OSSerialize* s,
                        OSCollection* container,
                        const OSSymbol* name,
                        const OSMetaClassBase* value);
```

Agenda

- New Mitigations in macOS Sierra
- Kernel Info Leak
- **Kernel Code Execution**
- Conclusion

Kernel Code Execution - Overview

- Accelerator is one of the devices that we can directly access in the webcontent sandbox

- The vulnerability lies in AMDRadeonXx000.kext (x may vary on different platforms)

- The vulnerability is an uninitialized stack variable issue and the exploit is quite straight-forward

Kernel Code Execution - Preparation

- Two different userclients
  - AMDSIGLContext (type 1)
  - AMDAccelSharedUserClient (type 6)
- Connect these two userclients
- One method call
  - AMDRadeonX4000_AMDSIGLContext::surfaceCopy (selector 0x201)

Kernel Code Execution - surfaceCopy()

- Local variable is not initialized to 0
- User can control the resource id to lookup
- LookupResource() not always initialize the local variable
- A vcall is invoked blindly on these local variables

```
IOAccelShared2::lookupResource(v5, a2[2], &v51);
IOAccelShared2::lookupResource(v5, a2[1], (void **)&v50);
v6 = -536870206;
if ( !v51 || !v50 )
  goto LABEL_42;
v12 = (*(__int64 (**)(void))(*(_QWORD *)v51 + 368LL))();
v13 = (*(__int64 (**)(void))(*(_QWORD *)v50 + 368LL))();
```

Kernel Code Execution - Control the Stack

- The uninitialized value on kernel stack is random so we need to control the stack

- Luckily, a function (selector 7333) in the AGPM userclient come to rescue

- We are able to copy at most 4096 bytes of controlled, non-zero data onto kernel stack

```
case 7333:
  kprintf("kAGPMSetPlimit plimit = %llu type = %s\n", *a2->scalarInput, a2->structureInput);
  v16 = (char *)&v22 - ((a2->structureInputSize + 1 + 15LL) & 0xFFFFFFFFFFFFFFF0LL);
  strncpy(v16, (const char *)a2->structureInput, a2->structureInputSize);
```

Kernel Code Execution - Value on Stack

- What value should we “initialize” for the uninitialized stack variable

- A pointer pointing to an object we fake on the heap

- A pointer pointing to a real object on the heap

  - It is hard to find such a candidate

  - You may have a try ;-)

Kernel Code Execution - Value on Stack

- How can we know where our fake objects locates?
- Option - 1
  - Reuse that leak to get heap address info
    - Hunt for appropriate object - HARD
    - Too many free-refill operations - UNSTABLE

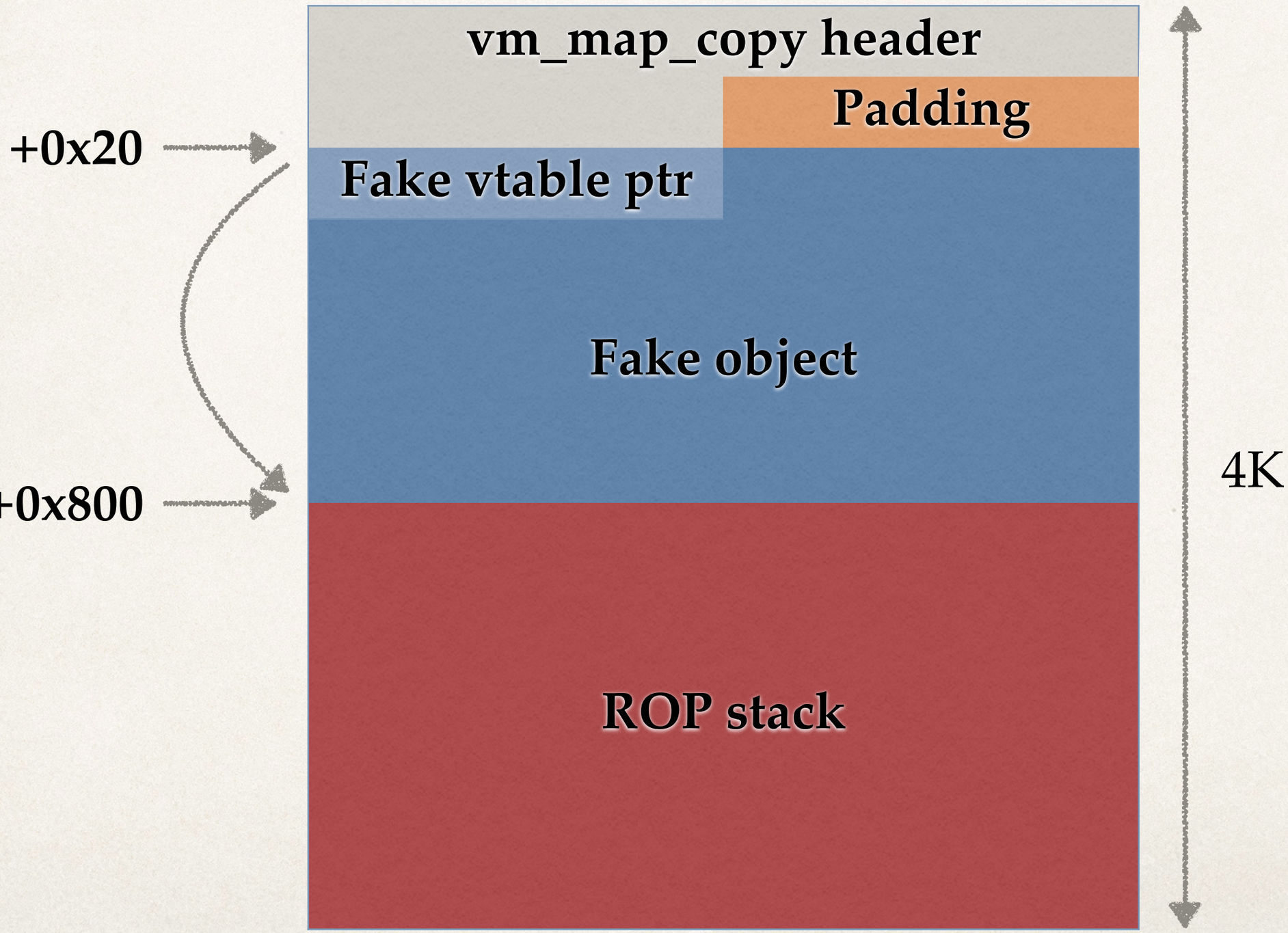
Kernel Code Execution - Value on Stack

- How can we know where our fake objects locates?
- Option - 2
  - Spray several GBs of data in the kernel
    - Heap randomization weak in the kernel
    - User-controlled data locates at fixed address

Kernel Code Execution - Spray Fake Objects

- Spray with vm_map_copy
  - Fast and small side effects on heap
  - Arbitrary controlled size (up to 4K) and contents
  - Uncontrolled beginning 0x18 bytes - NOT matter
  - Fake object contains a fake vtable and a fake stack

Kernel Code Execution - Spray Fake Objects



Kernel Code Execution - ROP Chain

- Save registers
- Pivot Stack
- Disable SMEP & SMAP
- Return to userland SHELLCODE
- Re-enable SMEP & SMAP
- Call _thread_exception_return() to exit

Kernel Code Execution - Disable SMEP&SMAP

- Bits stored in CR4 register indicate the states of CPU
- Clear 21st to disable SMEP and 22nd for SMAP
- gadgets:
  - read CR4: mov rax, cr4 ,…, ret
  - Write CR4: mov cr4, rax ,…, ret

Kernel Code Execution - Special Notes

- The kernel stack used when entering kernel mode varies from time to time
  - call sprayStack() several times to cover more stacks
- Do not forget to unlock the locks locked in AMDSIGLContext::surfaceCopy()

Agenda

- New Mitigations in macOS Sierra
- Kernel Info Leak
- Kernel Code Execution
- **Conclusion**

Conclusion

- New mitigations make exploit more challenging and encourage us to discover new exploit techniques
- Keep initialization in our mind when coding

Thank You For Your Attention

Q&A