Pwning the macOS Sierra Kernel inside the Safari Sandbox

Tielei Wang & Hao Xu

Team Pangu

Agenda

- Introduction
- Kernel Attack Surface in Safari
- Kernel Vulnerabilities and Exploitations
- Conclusion

About me

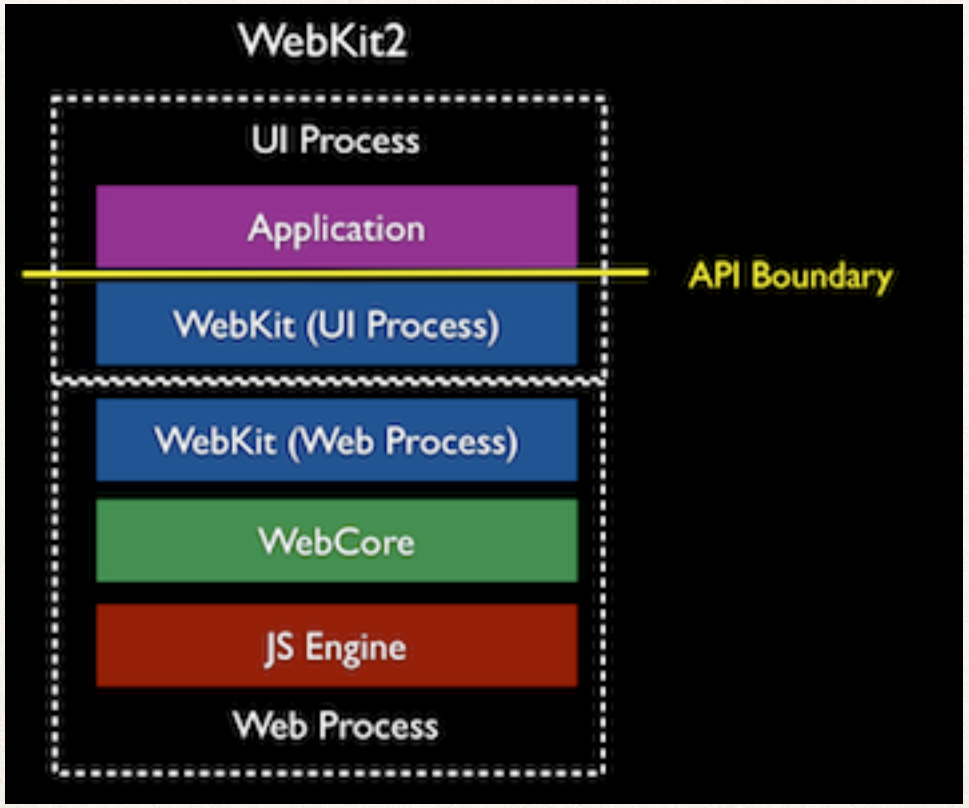
- Tielei Wang
  - Co-founders of Team Pangu
  - known for releasing jailbreak tools for iOS 7-9
  - present research at BlackHat, CanSecWest, POC, RuxCon, etc.

macOS Sierra

- macOS Sierra (version 10.12) is the thirteenth major release of macOS
- released to end users on September 20, 2016
- hot target in PWN contest
  - Pwn2own
  - PwnFest

Overview of Pwning Safari

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

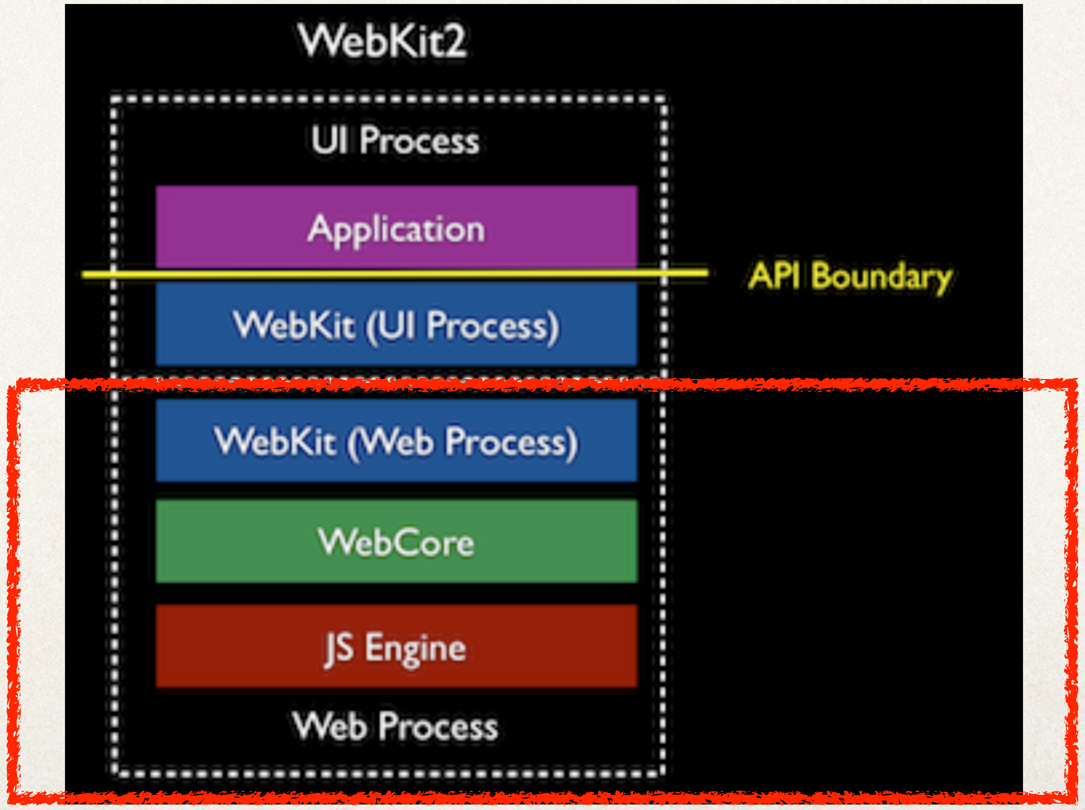


https://googleprojectzero.blogspot.co.id/2014/11/pwn4fun-spring-2014-safari-part-ii.html

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

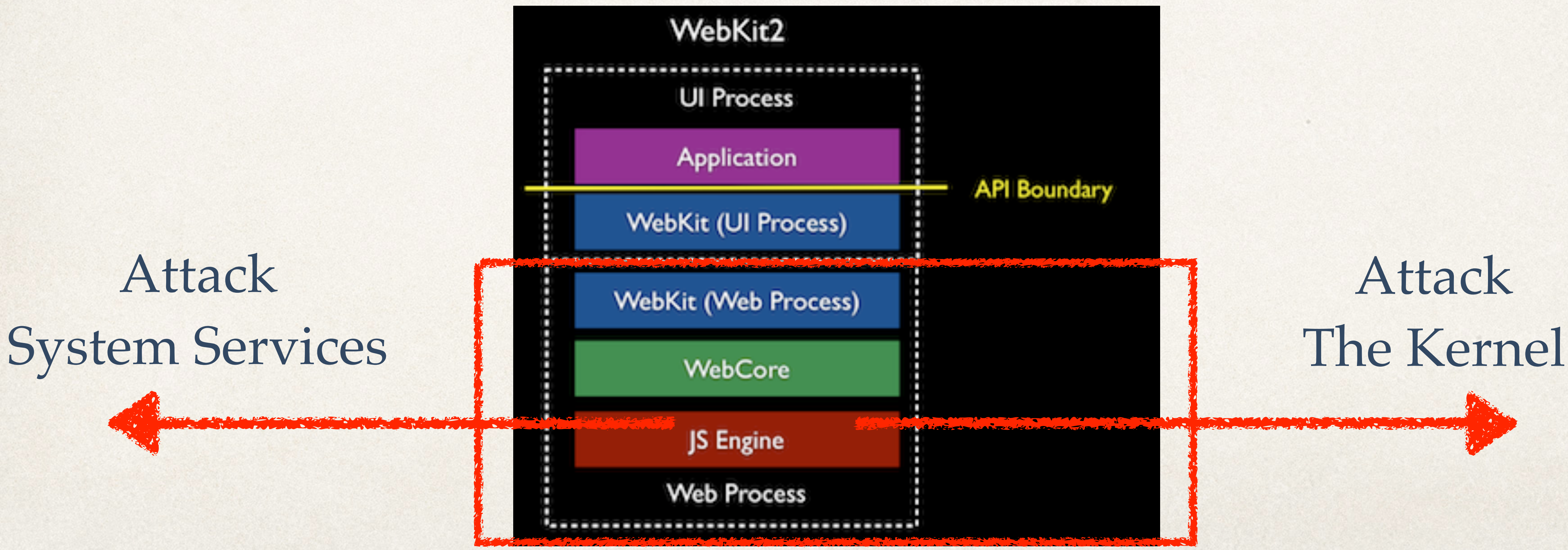
Overview of Pwning Safari

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



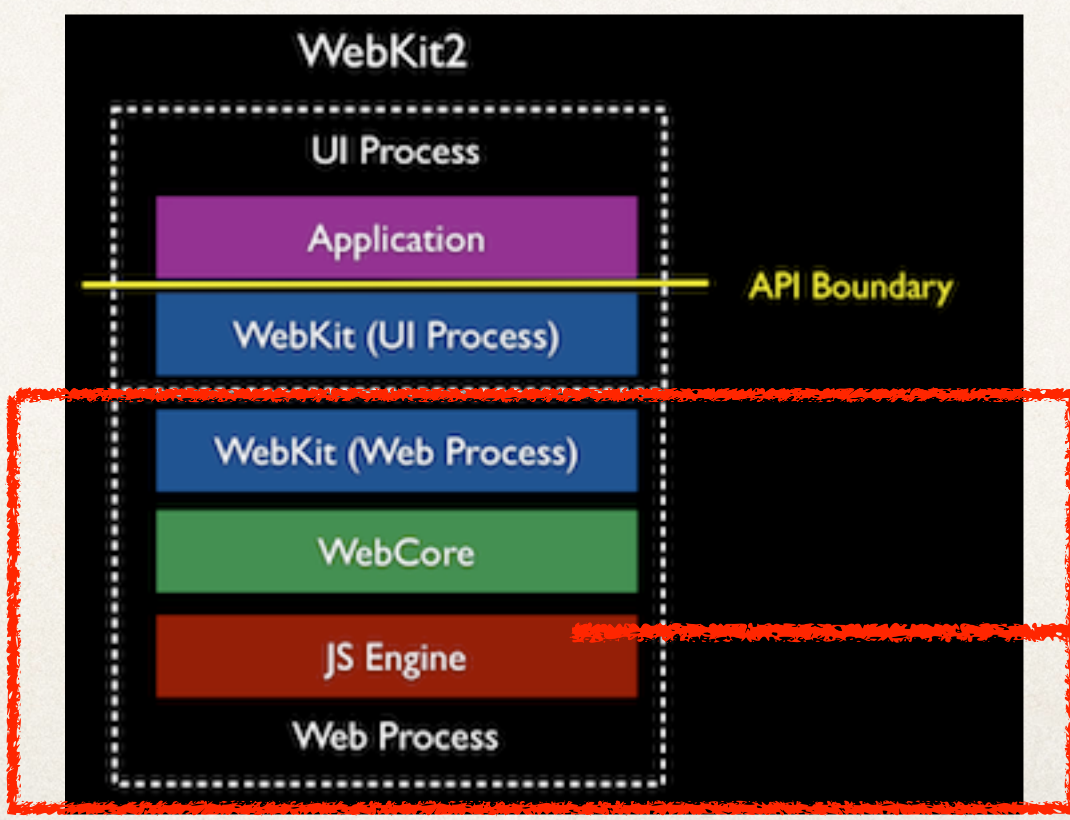
Overview of Pwning Safari

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



Focus of Our Talk

- Directly exploit kernel vulnerabilities



Attack
The Kernel

Agenda

- Introduction
- **Kernel Attack Surface in Safari**
- Kernel Vulnerabilities and Exploitations
- Conclusion

WebProcess Sandbox Profiles

SHAKACON
SUN, SURF, & C SHELLS

- /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"))
```

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

IOKit Interface = Attack Surface

- 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

Agenda

- Introduction
- Kernel Attack Surface in Safari
- **Kernel Vulnerabilities and Exploitations**
- Conclusion

Kernel Vulnerabilities

- Uninitialized heap in IOAudioFamily leading to info leak
- Uninitialized stack in AMD graphics driver leading to arbitrary code execution
- Heap overflow in IOAcceleratorFamily2 caused by double-fetch in shared memory leading to arbitrary code execution

Case 1. IOAudioFamily Info Leak

- The vulnerability lies in IOAudioFamily (<= 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

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 the kalloc.72 zone
- The *sender* field offset is 0x38
- 8-byte data at the offset of 0x38 from a freed object in kalloc.72 leaked

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 in webcontent process
- Challenge 2: How to leak critical information such as KALSR slide

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

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

Leak From 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;
    ...
}
```

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

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

Case 2. AMDRadeon Code Execution

- 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

Vulnerability Overview

- AMD Accelerator can create different user clients according to the type parameter

```
kern_return_t IOServiceOpen(io_service_t service, task_port_t owningTask, uint32_t type,
io_connect_t *connect);
```

- AMDSIGLContext (type 1)
- AMDAccelSharedUserClient (type 6)

Vulnerability Overview

- AMDSIGLContext is not started until AMDAccelSharedUserClient is connected

```
kern_return_t IOConnectAddClient(io_connect_t connect, io_connect_t client);
```

- AMDRadeonX4000_AMDSIGLContext's externalMethods interface dispatches functions according to method selector

```
kern_return_t IOConnectCallMethod(mach_port_t connection, uint32_t selector, …);
```

Vulnerability Overview

- selector 0x201 will reach function AMDRadeonX4000_AMDSIGLContext::surfaceCopy
- surfaceCopy is supposed to lookup a resource object according to input index and proceeds to use the resource object

```
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))();
```

Vulnerability Analysis

- v51 and v50 are two local variables on the stack
- lookupResource may fail for invalid index
- surfaceCopy neither initializes the local variables nor checks the return value from lookupResource
- Supplying invalid index will trigger a panic

```
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))();
```

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 comes 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);
```

Control The 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 ;-)

Control The 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

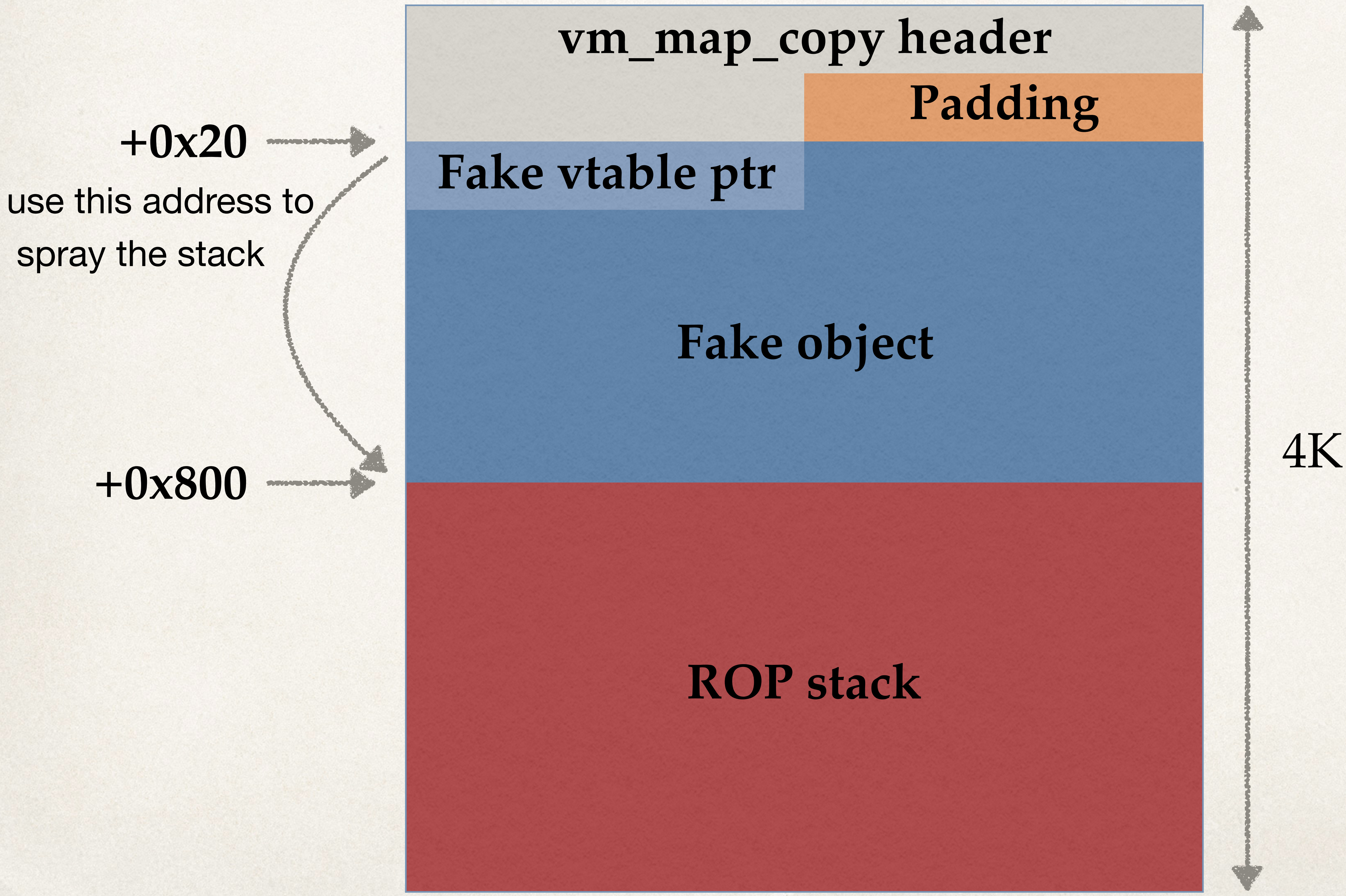
Control The Stack

- 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 a fixed address

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

Spray Fake Objects



ROP Chain

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

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

Special Notes

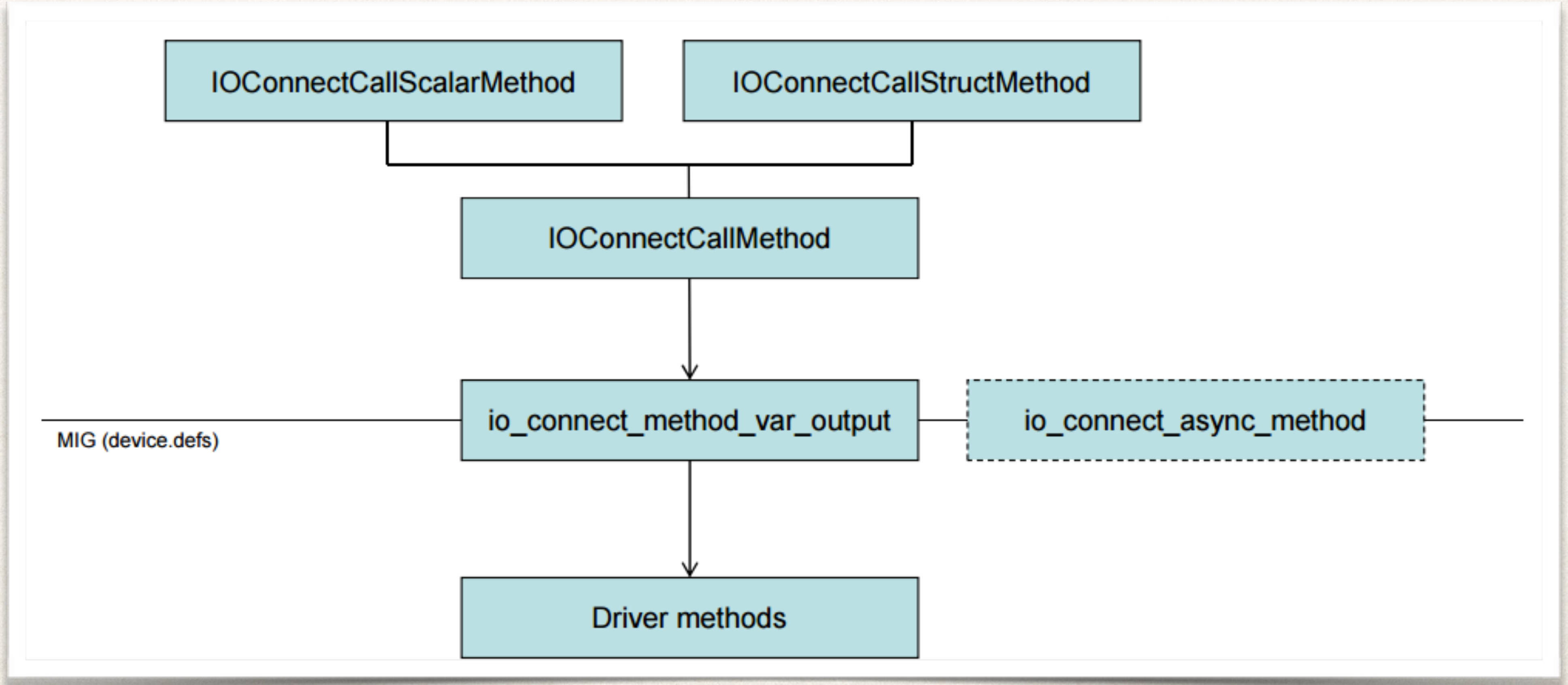
- The kernel stack used when entering kernel mode varies from time to time
  - call sprayStack() several times to cover more stacks
- Do not forget to unlock the locks locked in AMDSIGLContext::surfaceCopy()

Case 3. IOAcceleratorFamily2 Heap Overflow

- One of my favorite issue.
- It is caused by a fundamental issue in IOKit data sharing mechanism that results in many kernel bugs
- Reported (~~burned~~) by KeenLab to Apple

Implicit Shared Memory

- IOConnectCallMethod is used to send/receive data to/from an IOUserClient object



Implicit Shared Memory

```
/* Routine io_user_client_method */
kern_return_t is_io_connect_method
(
	io_connect_t connection,
	uint32_t selector,
	io_scalar_inband64_t scalar_input,
	mach_msg_type_number_t scalar_inputCnt,
	io_struct_inband_t inband_input,
	mach_msg_type_number_t inband_inputCnt,
	mach_vm_address_t ool_input,
	mach_vm_size_t ool_input_size,
	io_struct_inband_t inband_output,
	mach_msg_type_number_t *inband_outputCnt,
	io_scalar_inband64_t scalar_output,
	mach_msg_type_number_t *scalar_outputCnt,
	mach_vm_address_t ool_output,
	mach_vm_size_t *ool_output_size
)
```

up to 16 uint64_t

up to 4096 chars

userland address

- io_connect_method can pass shared memory to the kernel objects through ool_input

IOAccelDisplayPipeUserClient2

- IOGraphicsAccelerator2::newUserClient can return a userclient object of IOAccelDisplayPipeUserClient2 (type=4)
- IOAccelDisplayPipeUserClient2::externalmethod supports many methods
  - recall that io_connect_method will reach IOAccelDisplayPipeUserClient2::externalmethod

Vulnerability Analysis

- IOAccelDisplayPipePostCSCGammaVID::init

```
char __fastcall IOAccelDisplayPipePostCSCGammaVID::init(__int64 a1, __int64 ool_input)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = (void *)IOMalloc(*(_QWORD *)(ool_input + 40));
  *(_QWORD *)(a1 + 56) = v2;
  if ( !v2 )
    return 0;
  *(_DWORD *)(a1 + 12) = *(_DWORD *)(ool_input + 4);
  *(_QWORD *)(a1 + 16) = *(_QWORD *)(ool_input + 8);
  *(_QWORD *)(a1 + 24) = *(_QWORD *)(ool_input + 16);
  *(_QWORD *)(a1 + 32) = *(_QWORD *)(ool_input + 24);
  v3 = *(_QWORD *)(ool_input + 32);
  *(_DWORD *)(a1 + 40) = v3;
  *(_DWORD *)(a1 + 44) = HIDWORD(v3);
  size = *(_QWORD *)(ool_input + 40);
  *(_QWORD *)(a1 + 48) = size;
  memcpy(v2, (const void *)(ool_input + 48), size);
  return 1;
}
```

alloc heap with controllable size

refetch the size from shared memory

perfect heap overflow in memcpy

Vulnerability Exploitation

- Racing the double-fetch
- Thread1: send ool input to IOAccelDisplayPipeUserClient2
- Thread2: modify the size field

Vulnerability Exploitation

- Heap Overflow occurs in memcpy
- What we can control
  - The dest buffer's size
  - The src buffer's size (through race condition)
  - The src buffer's content (in shared memory)

Kernel Vulnerability Summary

- Case 1 and 2 were fixed in 10.12.3, January 23, 2017
  - CVE-2017-2358
  - CVE-2017-2357
  - We used the two vulns in PwnFest 2016
- Case 3
  - No public CVE
  - Apple changed ool shared memory as COW

Agenda

- Introduction
- Kernel Attack Surface in Safari
- Kernel Vulnerabilities and Exploitations
- **Conclusion**

Conclusion

- Kernel exploitation is fun
- Exploiting macOS kernel inside Safari sandbox is hard, but is feasible, and will continue to be feasible

Thank You For Your Attention

Q&A