Kernel Exploitation on Apple's M1 Chip

@08Tc3wBB | ZecOps Mobile EDR

#OBTS v4.0 | Maui, Hawaii, USA | Sept 30th, 2021

Special thanks to Zuk Avraham (@ihackbanme)

AppleAVE2

- It's a IOKit driver runs in kernel space
- Handles video encoding in formats: H264, HEVC, etc
- Only for ARM-based devices
  - iOS
  - iPadOS
  - M1 Chip Macs

AppleAVE2

- Before Apple introduces the M1 chip (Nov, 2020)
  - Only iOS
  - Closed source code, and most symbols have been deleted
- SBX 0day or jailbroken device is required to debug this driver
  - Less auditing eyes ;)

AppleAVE2

- Researcher Adam found a lot of vulnerabilities in this driver back in 2017
- Apple didn't bar access to AppleAVE2 from sandbox back then

CVE-2017-6998

An attacker can hijack kernel code execution due to a type confusion

CVE-2017-6994

An information disclosure vulnerability in the AppleAVE.kext kernel extension allows an attacker to leak the kernel address of any IOSurface object in the system.

CVE-2017-6989

A vulnerability in the AppleAVE.kext kernel extension allows an attacker to drop the refcount of any IOSurface object in the kernel.

CVE-2017-6997

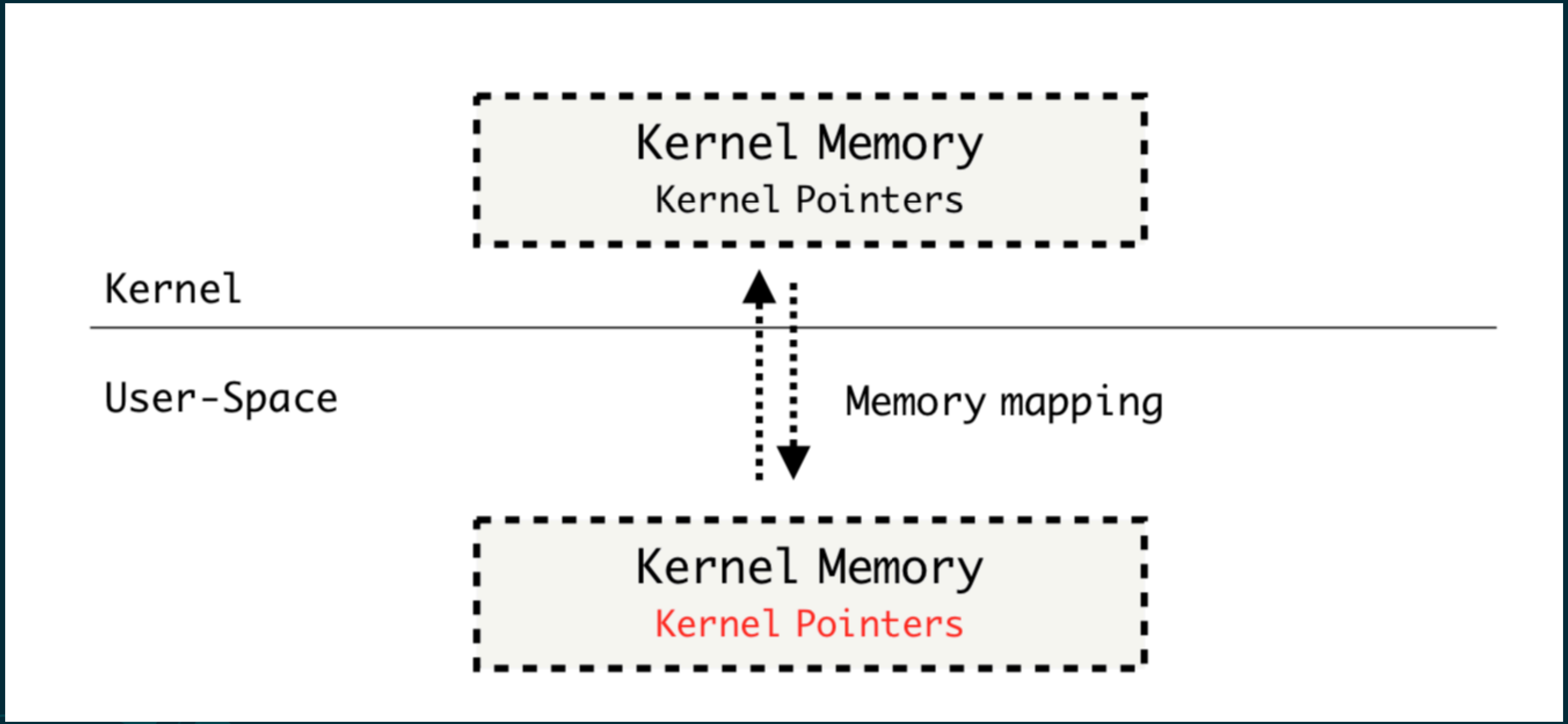
An attacker can free any pointer of size 0x28.

CVE-2017-6999

A user-controlled pointer is zeroed.

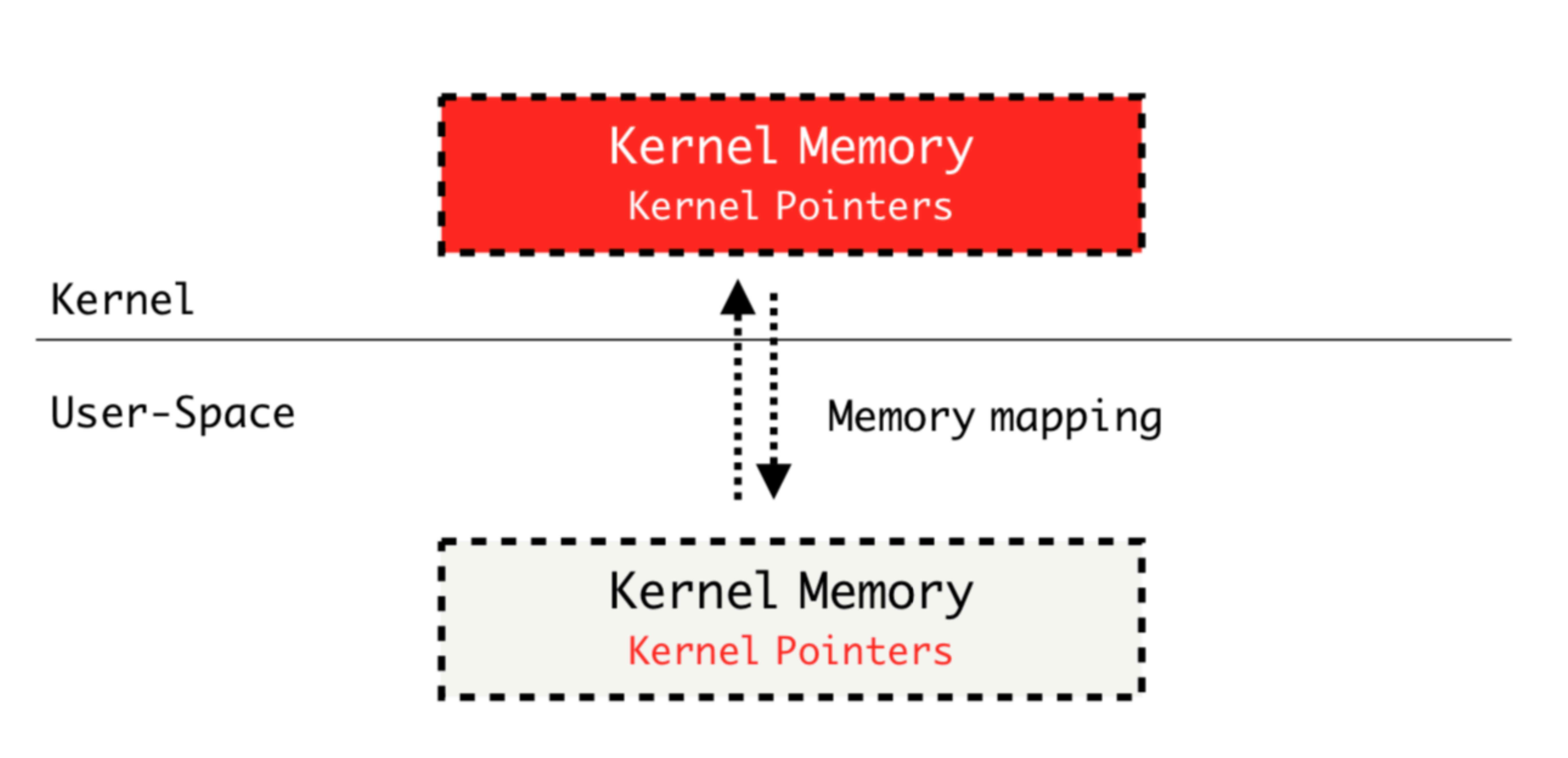
AppleAVE2

These vulnerabilities discovered by Adam in 2017 are very straightforward and easy to trigger



AppleAVE2 (2017)

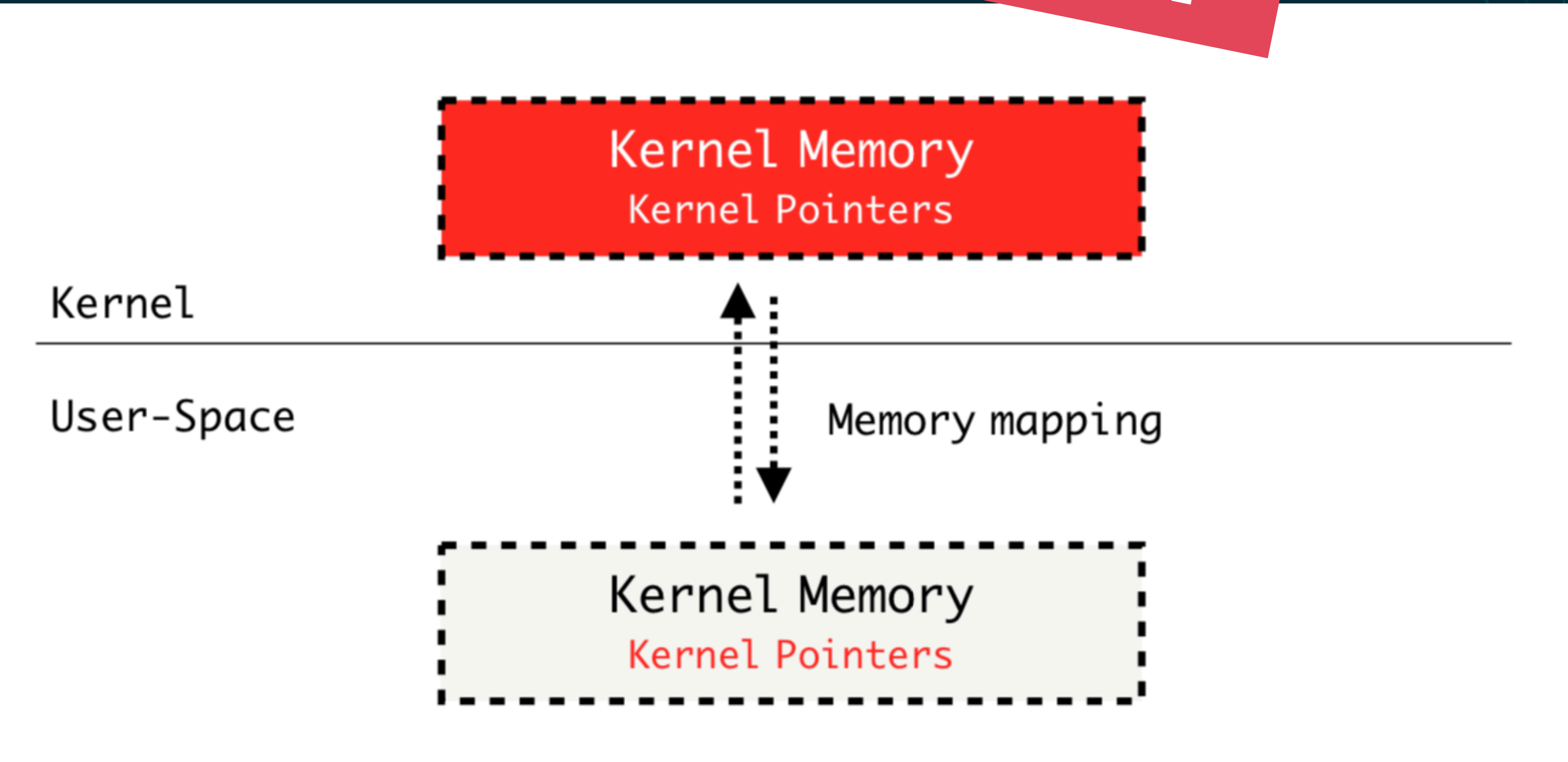
- Kernel Pointer Hijacking
  - Free arbitrary kernel memory
  - Empty arbitrary kernel memory
  - Arbitrary code execution on Non-PAC device
  - Race Conditions
- Kernel Pointer Leaking
  - Bypass KASLR
  - Assist Heap feng shui



AppleAVE2 (2017)

This is 2017!

- Kernel Pointer Hijacking
  - Free arbitrary kernel memory
  - Empty arbitrary kernel memory
  - Arbitrary code execution on Non-PAC device
  - Race Conditions
- Kernel Pointer Leaking
  - Bypass KASLR
  - Assist Heap feng shui



AppleAVE2 (2017)

In 2017, Apple "patched" bunch of AVE bugs

AVEVideoEncoder

Available for: iPhone 5 and later, iPad 4th generation and later, and iPod touch 6th generation

Impact: An application may be able to gain kernel privileges

Description: Multiple memory corruption issues were addressed with improved memory handling.

CVE-2017-6989: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6994: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6995: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6996: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6997: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6998: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

CVE-2017-6999: Adam Donenfeld (@doadam) of the Zimperium zLabs Team

Entry updated May 17, 2017

Adam Donenfeld | iOS | Jul 20 2017 |

As part of zLab's platform research team, I've tried to investigate an area of the kernel that wasn't thoroughly researched before. After digging into some of Apple's closed-source kernel modules, one code chunk led to another and I've noticed a little-known module, which I've never seen before, called AppleAVE.

AppleAVE was written neglecting basic security fundamentals, to the extent that the vulnerabilities described below were sufficient to pwn the kernel and gain arbitrary RW and root. Needless to say, due to the defragmentation of Apple's codebase for iOS, every iOS device running 10.3.1 or lower is currently vulnerable.

I've responsibly disclosed the vulnerabilities and Apple issued a security patch.

Apple's recent security patch that was shipped along with iOS 10.3.2, addresses 8 vulnerabilities I discovered: one vulnerability in the IOSurface kernel extension the other 7 in AppleAVEDriver.kext.

These vulnerabilities would allow elevation of privileges which ultimately can be used by the attacker to take complete control over affected devices.

AppleAVE2

- At first glance, this driver looks quite complicated to me.
  - I estimate that it's gonna take a week of reverse engineering work to learn the internal and write testing code.
    - “Apple must have reinforced this driver to a very secured level after Adam's discovery”
      - "So yeah, it's not worth spending a week on this"

AppleAVE2

- Apple is a company ran and operated by people
  - Someone who works for Apple read our report and did the patching work
    - Sometimes people are lazy, we don't want to put effort beyond necessary
      - Especially when effort is not being appreciated

AppleAVE2

- Apple is a company ran and operated by people
  - Someone who works for Apple read our report and did the patching work
    - Sometimes people are lazy, we don't want to put effort beyond necessary
      - Especially when effort is not being appreciated

Who found the bug?

Adam Donenfeld 😊

Who fixed the bug?

??? 😶

AppleAVE2

- Fix Solutions
  1. Simply block the access from app to this driver
  2. Carefully inspect any code interactive with the mapped memory; Design security mechanisms to Counter-Exploitation

AppleAVE2

1. Simply block the access from app to this driver

Get the job done effortlessly!

Shift security responsibility to sandbox

---

2. Carefully inspect any code interactive with the mapped memory; Design security mechanisms to Counter-Exploitation

Extra effort is Thankless!

Always check how the bug has been fixed

- Especially when you are the person who submitted the report!! You know this vulnerability inside and out.
- You can get huge return for being little more responsible!

zecOps

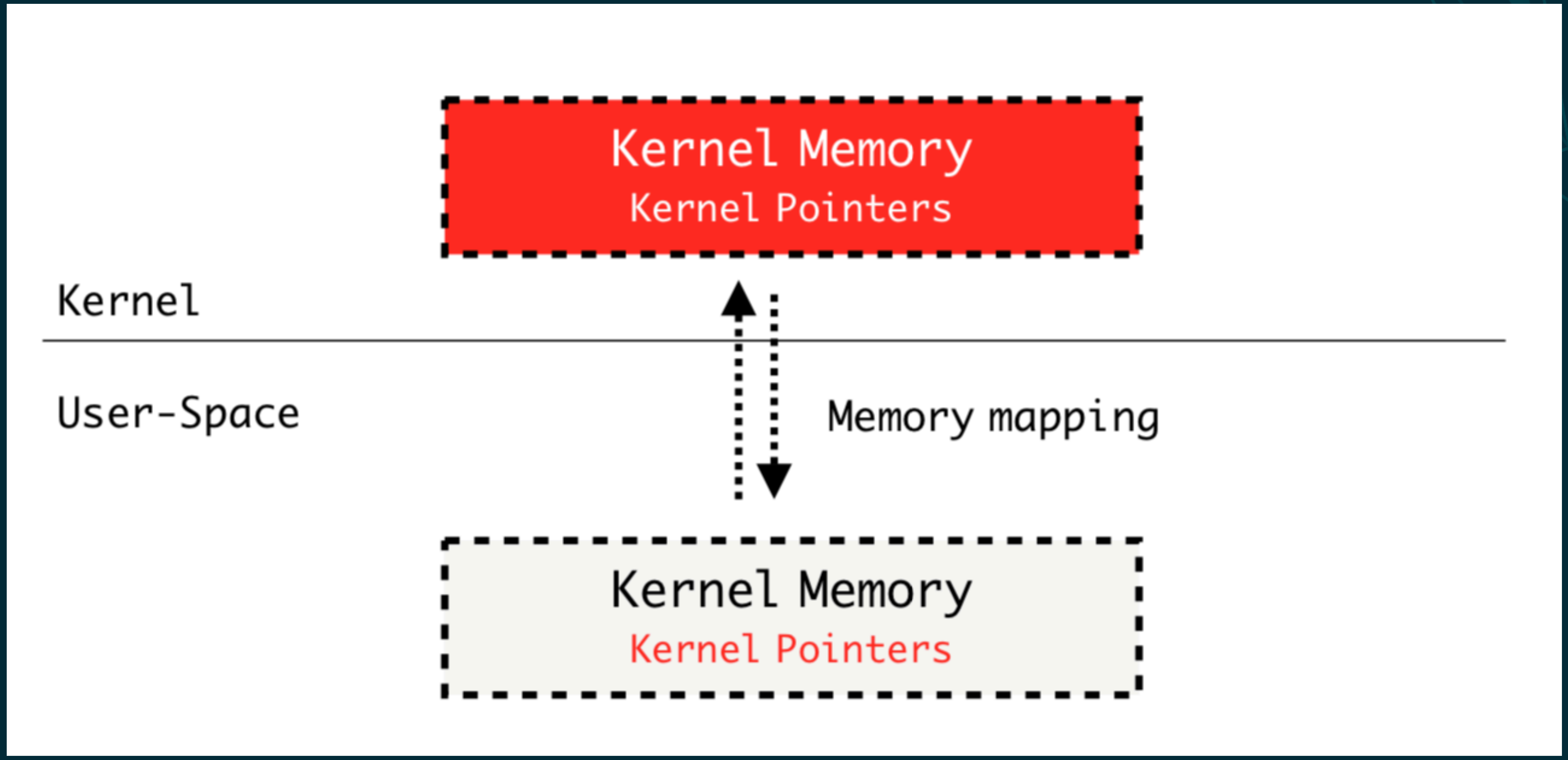
AppleAVE2 (2019)

Kernel Pointer Hijacking

- ✔ Free arbitrary kernel memory
- ✔ Empty arbitrary kernel memory
- ✔ Arbitrary code execution on Non-PAC device
- ✔ Race Conditions

Kernel Pointer Leaking

- ✔ Bypass KASLR
- ✔ Assist Heap feng shui



This is why we love Sandbox Escape

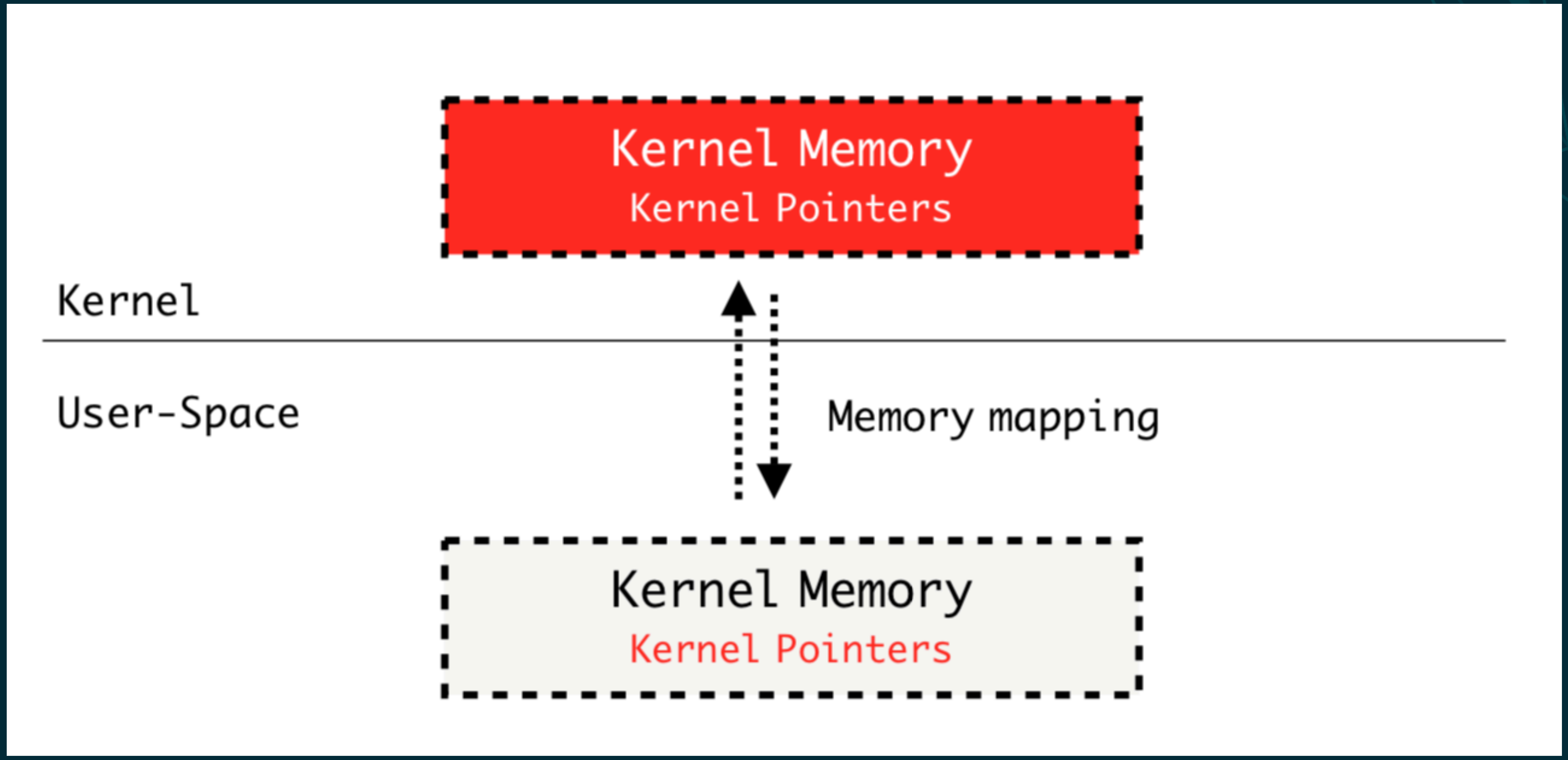
AppleAVE2 (2020)

Kernel Pointer Hijacking

- ✘ Free arbitrary kernel memory
- ✔ Empty arbitrary kernel memory
- ✘ Arbitrary code execution on Non-PAC device
- ✔ Race Conditions

Kernel Pointer Leaking

- ✔ Bypass KASLR
- ✔ Assist Heap feng shui



Jailbreak iOS 13
$100k Apple Security Bounty

AppleAVE2 (2021)

- Apple introduced AppleAVE2 on ARM-based macOS
  - Quite a lot changes

  - It's fully symbolized
    - Ease reverse engineering work A TON!

  - Note that sandbox is not mandatory on macOS
    - We can access AppleAVE2 directly!

AppleAVE2 (2021)

- A big trunk of code deal with *FrameInfo->InfoType* moved
  - From *AppleAVE2UserClient::SetSessionSettings*
  - To *AppleAVE2Driver::EnqueueGated*

- Introduce doubly linked list to manage clientbuf objects (AVE_DLList_*)
  - Perhaps it was meant to mitigate a technique I used on 13.7 Jailbreak — hijacking clientbuf structure

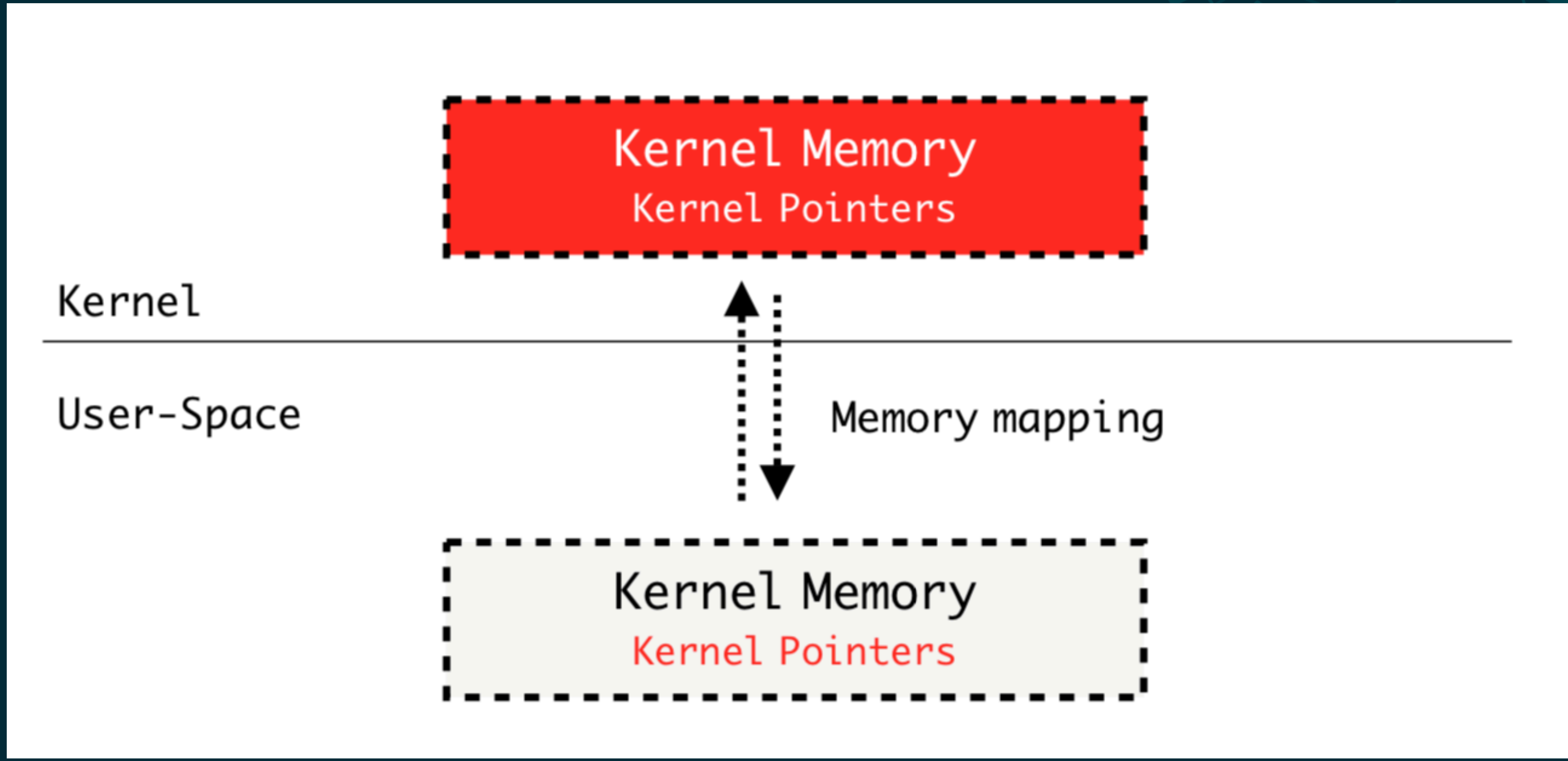
AppleAVE2 (2021)

Kernel Pointer Hijacking

- ✘ Free arbitrary kernel memory
- ✘ Empty arbitrary kernel memory
- ✘ Arbitrary code execution on Non-PAC device
- ✔ Race Conditions

Kernel Pointer Leaking

- ✘ Bypass KASLR
- ✘ Assist Heap feng shui



- ✔ Kernel R/W Primitives
- ✔ Bypass KASLR

AppleAVE2 (2021)

- Introduce doubly linked list to manage clientbuf objects (AVE_DLList_*)
  - Lots of new code
    - Provides new primitives that are powerful enough to achieve kernel R/W and bypass KASLR

AppleAVE2 (2021)

- The first vulnerability that caused memory corruption and led to the formation of kernel read primitives
  - The trigger path:

```
AppleAVE2UserClient::externalMethod
  -> AppleAVE2UserClient::_SetSessionSettings
    -> AppleAVE2UserClient::SetSessionSettings
      -> AppleAVE2Driver::Enqueue
        -> AppleAVE2Driver::EnqueueGated
          -> AppleAVE2Driver::Board
            -> AppleAVE2Driver::ProcessReady
              -> AppleAVE2Driver::ProcessReadyCmd
                -> AppleAVE2Driver::EncodeFrame
```

- The actual vulnerability is located in AppleAVE2Driver::EncodeFrame

zecOps

AppleAVE2 (2021)

- The actual vulnerability is located in AppleAVE2Driver::EncodeFrame

```
{
  ...
  while ( 1 )
    {
      v27 = *v25;
      if ( !*(_BYTE *)(v10 + v27 + 432) )
        break;
      v28 = *(signed int *)(userKernel_sharedMapping + 168);    // (1)
      *(_QWORD *)(clientbuf + 8 * v28 + 158920) = userKernel_sharedMapping; // (2)
      *(_DWORD *)(4 * v28 + 158920 + clientbuf + 136) = 2;
      ++*v26;
      *(_QWORD *)(userKernel_sharedMapping + 5976) = v24;

      v29 = AppleAVE2Driver::IMG_V_EncodeAndSendFrame(
                      v10,
                      (clientbuf *)clientbuf,
                      userKernel_sharedMapping,
                      (uint64_t *)(userKernel_sharedMapping + 5976));
   ...
}
```

- (1) v28 was read from a user-kernel shared mapping memory. The attacker could give v28 any value due to lack of size or overflow checks.
- (2) Then v28 is used as a vital offset to overwrite a specific location in clientbuf, because there is no size or overflow checks. The attacker could insert the userKernel_sharedMapping pointer to any location of clientbuf by controlling the value of v28

- How the kernel read primitive were built

```
There is a function called AppleAVE2Driver::ProcessReady in the vulnerability trigger path:

...
  v9 = &clientbuf->cmd_nodeList;
  v10 = AVE_DLList_Front(&clientbuf->cmd_nodeList);
  if ( !v10 )
    return 0;
  v16 = v10;
  do
  {
    if ( clientbuf->flag_skipCmd )
    {
      AppleAVE2Driver::SkipCmd(v8, clientbuf, v16, v11, v12, v13, v14, v15, v18, v19, v20, SHIDWORD(v20), v21);
    }
    else
    {
      if ( *(_DWORD *)&clientbuf->pad7[29] >= *(_DWORD *)&clientbuf->pad7[25] )
        return 0;
      AppleAVE2Driver::ProcessReadyCmd((__int64)v8, clientbuf, v16);   // (1)
    }
    AVE_DLList_PopFront(v9);
    AVE_BlkPool::Free(*(AVE_BlkPool **)&clientbuf->pad4[40], v16);
    v16 = (cmdbuf *)AVE_DLList_Front(v9);
  }
  while ( v16 );
...
```

- (1) The memory corruption occurrence happened in AppleAVE2Driver::ProcessReadyCmd, which allows us to insert a pointer into anything that's in range of clientbuf. The pointer points to a kernel memory that's mapped into the userspace, and we can control and modify its content anytime. We leverage this capability to overwrite clientbuf->cmd_nodeList pointer, directly control the value of v16, then in the next iteration, v16 gets pass to AppleAVE2Driver::ProcessReadyCmd

- How the kernel read primitive were built

```
AppleAVE2Driver::ProcessReadyCmd( this, clientbuf, v16 ):
{
  ...
  contorl_v = *(_QWORD *)(v16 + 48);
  ...
  result = AppleAVE2Driver::PreInitCreateContext(0LL, clientbuf, contorl_v);
  ...
}
```

```
AppleAVE2Driver::PreInitCreateContext // Then read 4 bytes off of contorl_v and send it to userland process:
{
  ...
  AppleAVE2UserClient::SendFrame(
      (AppleAVE2UserClient *)v9->connected_userClient,
      *(_DWORD *)(contorl_v + 4),
      0xCDCDCDCD,
      0LL,
      *(unsigned int *)(contorl_v + 24),
      0LL);
  ...

}
```

- Kernel Read Primitive: AppleAVE2Driver::PreInitCreateContext read 4 bytes off of contorl_v and send it to our userland process
- The triggering of this vulnerability happens in the function Trigger_AppleAVE2_Vuln_Overwriting_ptr() as part of my exploit code

AppleAVE2 (2021)

- The second vulnerability allows us to write a pointer into any kernel address
  - The trigger path:

```
AppleAVE2UserClient::externalMethod
  -> AppleAVE2UserClient::_Close
    -> AppleAVE2UserClient::Close
      -> AppleAVE2Driver::close
        -> AppleAVE2Driver::closeGated
          -> AppleAVE2Driver::AVE_DestroyContext
            -> AVE_SurfaceMgr::DestroySurface
              -> AVE_DLList_Erase
```

- The actual vulnerability is located in AppleAVE2Driver::AVE_DestroyContext

- The second vulnerability allows us to write a pointer into any kernel address

```
--- In AppleAVE2Driver::AVE_DestroyContext:
{
  ...
  do
  {
    userKernel_sharedMapping = KernelFrameQueue::getRequestedSpot((KernelFrameQueue *)v10, v12);
    v14 = *(_QWORD *)(userKernel_sharedMapping + 5976);
    if ( v14 )
    {
      AVE_SurfaceMgr::DestroySurface(*(_QWORD *)&clientbuf->pad[20], v14);   // (1)
      *(_QWORD *)(userKernel_sharedMapping + 5976) = 0;
    }
    ++v12;
    v10 = *(_QWORD *)(v9 + 112);
  }
  ...
}
```

- (1) The value of v14 was read from userKernel_sharedMapping, we can pass any value to v14 and result in calling

- The second vulnerability allows us to write a pointer into any kernel address

```
--- In AVE_SurfaceMgr::DestroySurface (this, v14)
{
  ...
  v9 = (struct psNode *)AVE_Surface::GetMgrNode((AVE_Surface *) v14);
  AVE_DLList_Erase(v9); // v9 is under our control
  ...
}
```

```
--- Proceed to AVE_DLList_Erase (struct psNode *a1)
{
  if ( !a1 )
  {
    ...
    panic("\"psNode != NULL\"");
  }
  v6 = a1->psNode_prev;
  if ( !a1->psNode_prev )
  {
    ...
    panic("\"psNode->psPrev != NULL\"");
  }
  v7 = a1->psNode_next;
  if ( v7 )
  {
    v6->psNode_next = v7;
    a1->psNode_next->psNode_prev = v6;   // (2)
    return;
  }
}
```

- (2) If we manage to get a1 point to a kernel memory that we have control over its content, we can form an arbitrary kernel write primitive with this line of code
- The triggering of this vulnerability happens in the function remove_client2() as part of my exploit code

Bug Fix

- Technically, they are not “fixed”
  - Apple did not take action on the overflow problem

- New functions:
  - AVE_CopyFrameInfoFromEx
  - AVE_CopyFrameInfoToEx

```
    }
    memcpy((void *)(v33 + v26), (const void *)(v33 + 22504), 0x11820uLL);
    memcpy((char *)this_1->current_clientbuf + v27, (char *)this_1->current_clientbuf + 94216, 0x1EDCuLL);
    memcpy((char *)this_1->current_clientbuf + v28, (char *)this_1->current_clientbuf + 102116, 0x259CuLL);
    memcpy((char *)this_1->current_clientbuf + v29, (char *)this_1->current_clientbuf + 111744, 0x5164uLL);
    v39 = (char *)this_1->current_clientbuf;
    v36 = &v39[v30];
    v37 = v39 + 132580;
    v38 = 26132LL;
    goto LABEL_22;
  }
LABEL_24:
  v40 = this_1->provider;
  v41 = *(_DWORD *)(v46 + 4);
  __asm { AUTIBSP }
  if ( (_B8 ^ 2 * _B8) & 0x4000000000000000LL )
    __break(0xC471u);
  return AppleAVE2Driver::Enqueue(v40, (IOService *)this_1, v41, (void *)v46);
}
```

macOS Big Sur 11.1   AppleAVE2UserClient::PreInit

That should solve the race condition problem

```
    }
    memcpy(&v32[v27], v32 + 5698, 0x11820uLL);
    memcpy((char *)v10->current_clientbuf + v28, (char *)v10->current_clientbuf + 94504, 0x1EDCuLL);
    memcpy((char *)v10->current_clientbuf + v29, (char *)v10->current_clientbuf + 102404, 0x259CuLL);
    memcpy((char *)v10->current_clientbuf + v30, (char *)v10->current_clientbuf + 112032, 0x5164uLL);
    v38 = (char *)v10->current_clientbuf;
    v35 = &v38[v31];
    v36 = v38 + 132868;
    v37 = 26132LL;
    goto LABEL_22;
  }
LABEL_24:
  AVE_CopyFrameInfoFromEx((__int64)v22, *(unsigned int *)(v45 + 4));
  v39 = v10->provider;
  v40 = *(unsigned int *)(v45 + 4);
  __asm { AUTIBSP }
  if ( (_B8 ^ 2 * _B8) & 0x4000000000000000LL )
    __break(0xC471u);
  return AppleAVE2Driver::Enqueue((__int64)v39);
}
```

macOS Big Sur 11.4   AppleAVE2UserClient::PreInit

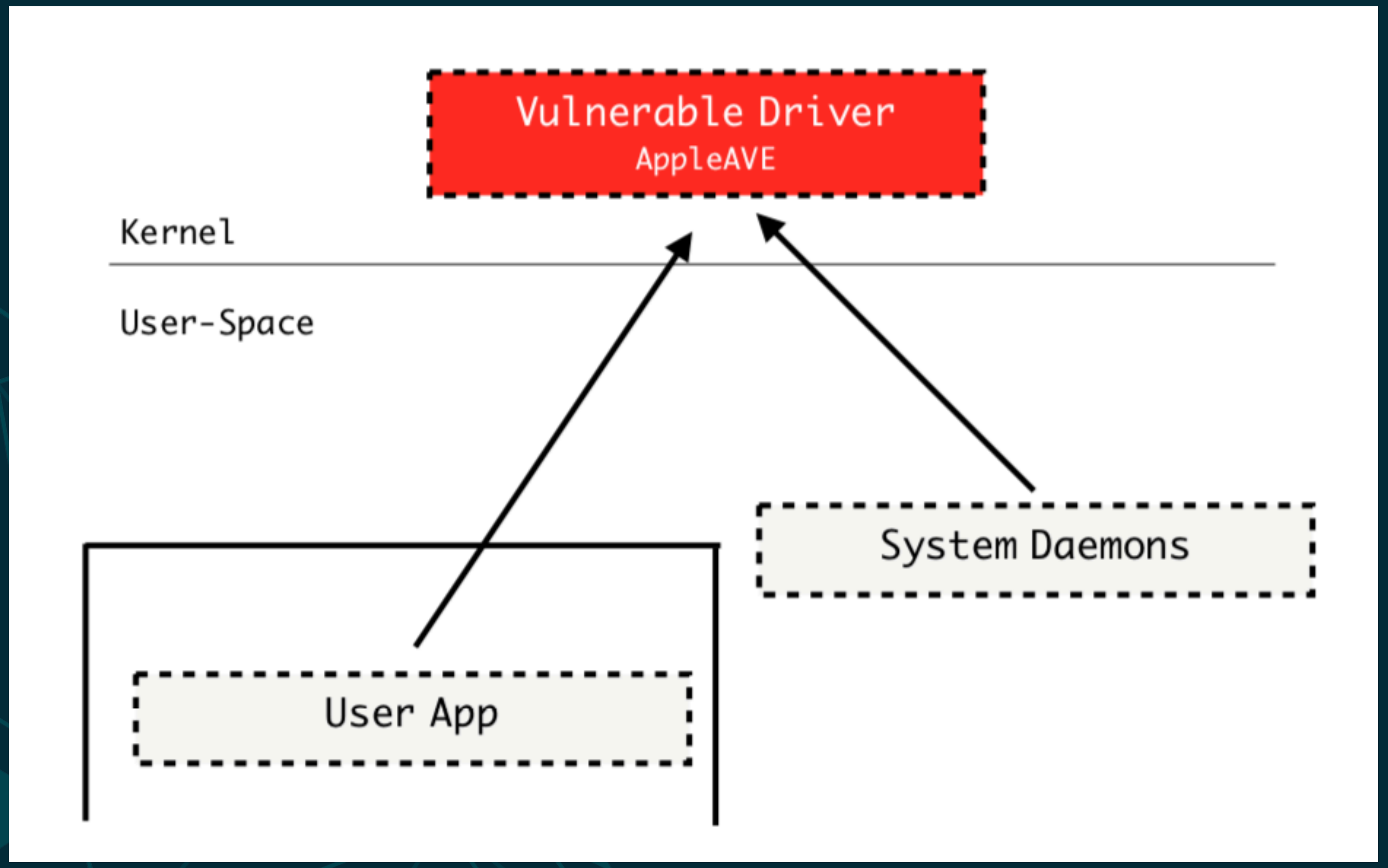
Apple Security Bounty

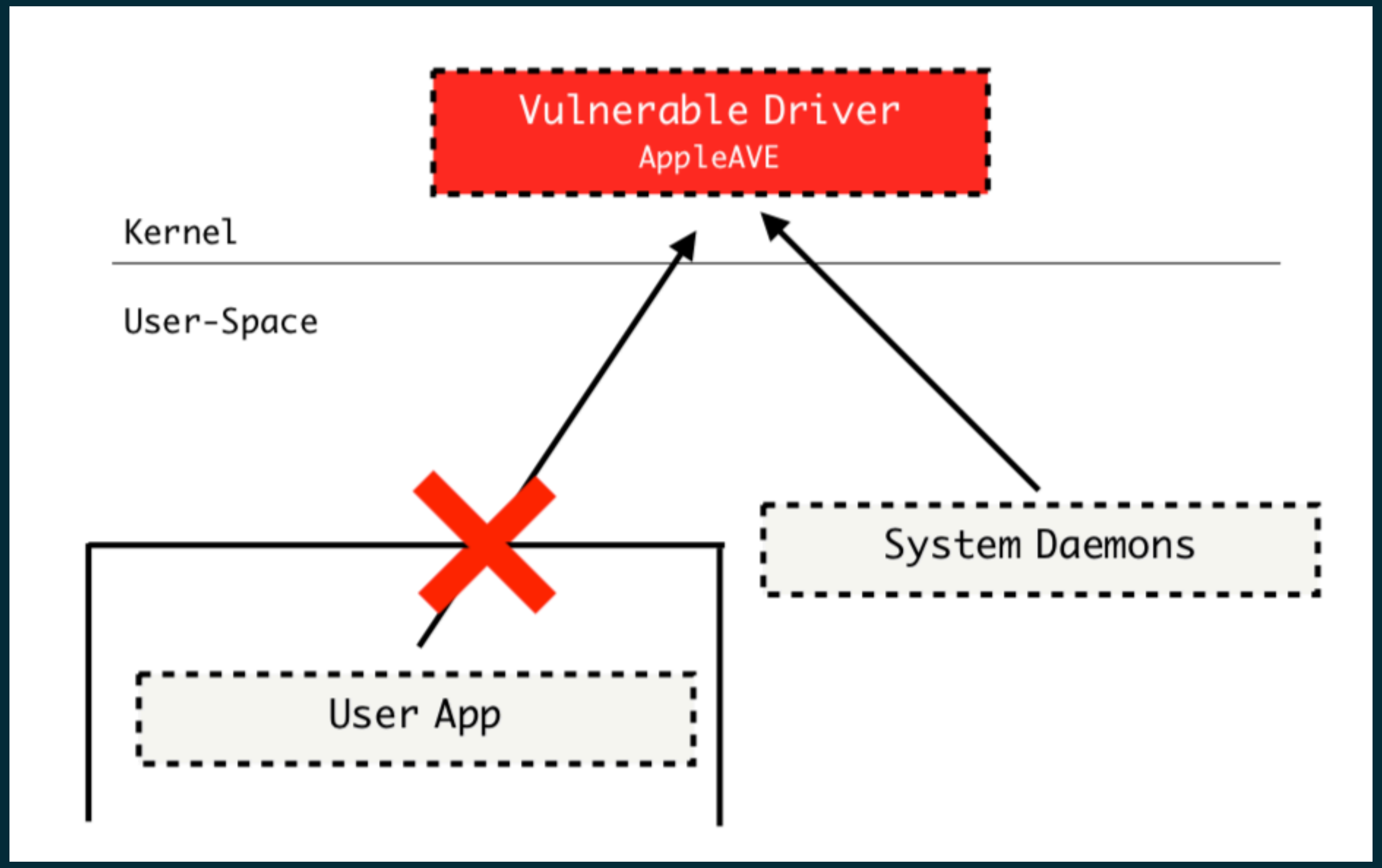
- I reported it in February, 2021
- The submission includes
  - Detailed technical description of the vulnerability
  - A proof-of-concept exploit that can get you a root shell
- Apple decided to award me $52,500

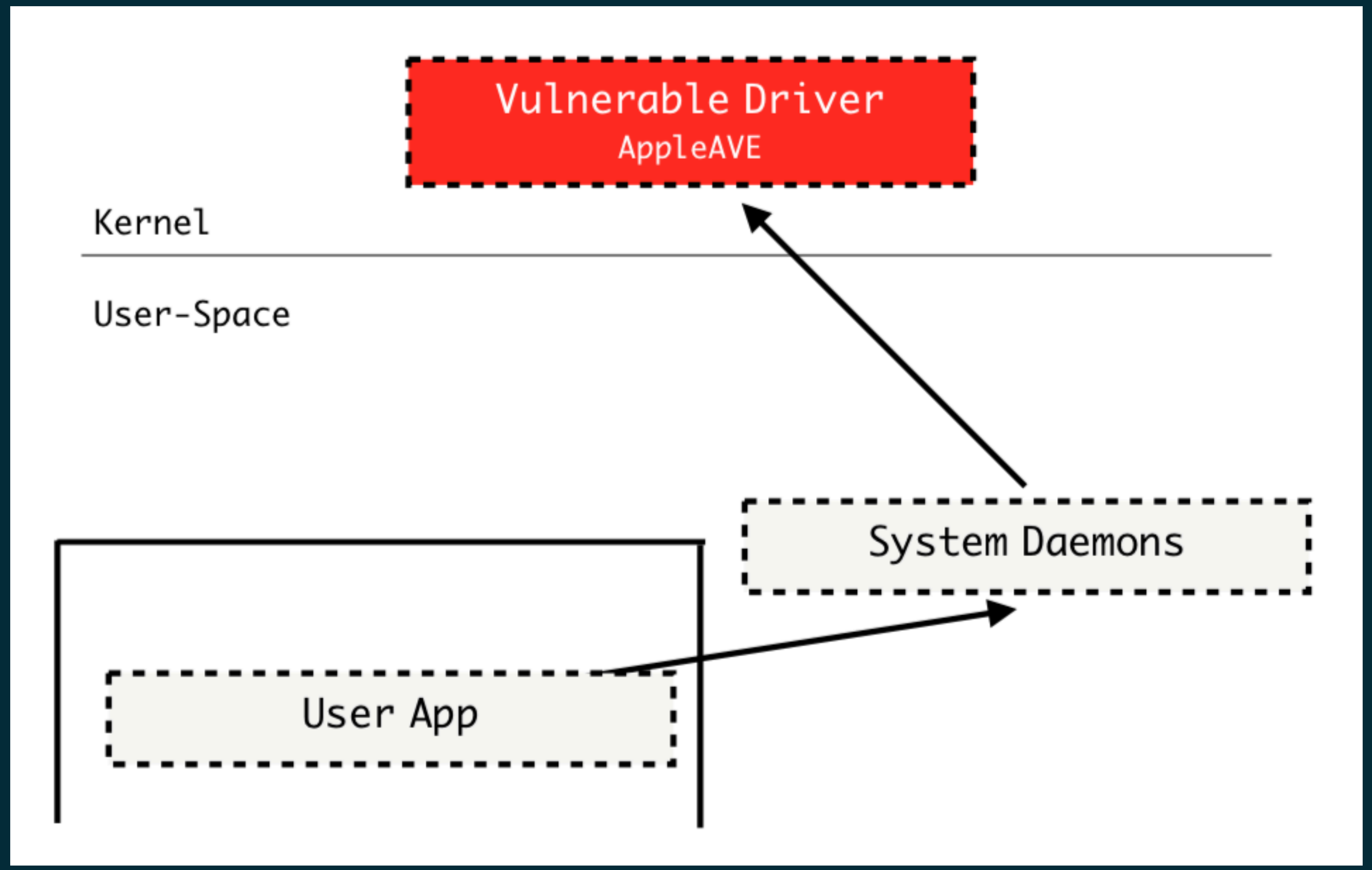
- Apple is being generous

Sandbox

- A simpler solution for patching a vulnerability
  - Block the access from sandbox
    - Shift security responsibility to sandbox







Negligence Outside Sandbox

- Back then, security outside of sandbox often got overlooked
  - Maybe it still is now, it's hard to tell
    - Our perception is limited by the time we are living in

zecOps

```
__int64 __fastcall ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment(__int64 a1, unsigned int *a2, __int64 a3, __int6
{
  __int64 v8; // x0
  __int64 v9; // x19

  if ( a2 )
  {
    v8 = (*(__int64 (__fastcall **)(_QWORD, _QWORD, _QWORD, char *, __int64, char *, _QWORD, __int64))(**(_QWORD **)(a
           *(_QWORD *)(a1 + 216),
           *a2,
           *((unsigned __int16 *)a2 + 2),
           (char *)a2 + 6,
           a3,
           (char *)a2 + 54,
           a2[30],
           a8);
    v9 = v8;
    if ( (_DWORD)v8 )
      IOLog(
        "[ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment] ProvInfoIOKit::getEncryptedSeedSegment returned %d\n",
        v8);
  }
  else
  {
    IOLog("[ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment] Error: null pointer for input structure\n");
    v9 = 0xE00002C2LL;
  }
  return v9;
}
```

```
__int64 __fastcall ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment(__int64 a1, unsigned int *a2, __int64 a3,
{
  __int64 v8; // x19
  char *v9; // x0
  __int64 v10; // x0
  __int64 v12; // [xsp+0h] [xbp-20h]

  if ( !a2 )
  {
    v8 = 0xE00002C2LL;
    v9 = "[ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment] Error: null pointer for input structure\n";
    goto LABEL_7;
  }
  if ( a2[30] >= 0x41 )
  {
    v8 = 0xE00002C2LL;
    v9 = "[ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment] Error: bad input structure lengths\n";
LABEL_7:
    IOLog(v9, v12);
    return v8;
  }
  v10 = (*(__int64 (__fastcall **)(_QWORD, _QWORD, _QWORD, char *, __int64, char *))(**(_QWORD **)(a1 + 216) +
          *(_QWORD *)(a1 + 216),
          *a2,
          *((unsigned __int16 *)a2 + 2),
          (char *)a2 + 6,
          a3,
          (char *)a2 + 54);
  v8 = v10;
  if ( (_DWORD)v10 )
  {
    v12 = v10;
    v9 = "[ProvInfoIOKitUserClient::ucGetEncryptedSeedSegment] ProvInfoIOKit::getEncryptedSeedSegment returned
    goto LABEL_7;
  }
  return v8;
```

The vulnerability is that the size argument to `memmove` is completely attacker controlled and not checked. This leads to kernel heap corruption.

CVE-2019-7287

- Missing size check when processing input data in ProvInfoIOKitUserClient

According to GP0, this was exploited in-the-wild combined with a SBX (CVE-2019-7286)

Reference: https://www.antid0te.com/blog/19-02-23-ios-kernel-cve-2019-7287-memory-corruption-vulnerability.html

My checklist for drivers that cannot be reached from inside the sandbox, at the time of iOS 12.

```
deny(1) iokit-open AUCUserClient // BAD!
deny(1) iokit-open AppleAOPAudioUserClient // BAD!
deny(1) iokit-open AppleAOPVoiceTriggerUserClient // BAD!
deny(1) iokit-open AppleAPFSUserClient
deny(1) iokit-open AppleAVE2UserClient // Wow!
deny(1) iokit-open AppleBasebandUserClient // BAD! Unsupported/Unimp
deny(1) iokit-open AppleCredentialManagerUserClient
deny(1) iokit-open AppleEffaceableStorageUserClient // BAD! Require Root
deny(1) iokit-open AppleFirmwareUpdateUserClient // BAD! Require entitlement
deny(1) iokit-open AppleFirmwareUpdateUserClient // BAD! Require entitlement
deny(1) iokit-open AppleHIDTransportBootloaderUserClient // BAD! Require entitlement
deny(1) iokit-open AppleHIDTransportDeviceUserClient // BAD! Require entitlement
deny(1) iokit-open AppleHIDTransportInterfaceUserClient // BAD! Require entitlement
deny(1) iokit-open AppleMobileApNonceUserClient // BAD! Require root
deny(1) iokit-open AppleMobileFileIntegrityUserClient
deny(1) iokit-open AppleNVMeUpdateUC
deny(1) iokit-open ApplePMPUserClient // BAD! Require root
deny(1) iokit-open ApplePPMUserClient // Analyzing
deny(1) iokit-open AppleSMCClient
deny(1) iokit-open AppleSMCWirelessChargerUserClient // Analyzing
deny(1) iokit-open AppleSPUAppDriverUserClient // BAD!
deny(1) iokit-open AppleSPUHapticsAudioUC // BAD!
deny(1) iokit-open AppleSPUProfileDriverUserClient // Wow! Info Leak
deny(1) iokit-open AppleSPUUserClient // BAD!
deny(1) iokit-open AppleStockholmControlUserClient // BAD! Too little stuff
deny(1) iokit-open IOAESAcceleratorUserClient
deny(1) iokit-open IOAccessoryIDBusUserClient // BAD!
deny(1) iokit-open IOAccessoryManagerUserClient // Analyzing
deny(1) iokit-open IOAudioCodecsUserClient
deny(1) iokit-open IODARTMapperClient // Analyzing
deny(1) iokit-open IOReportUserClient
deny(1) iokit-open IOTimeSyncClockManagerUserClient
deny(1) iokit-open IOTimeSyncDomainUserClient
deny(1) iokit-open IOTimeSyncgPTPManagerUserClient
deny(1) iokit-open IOUSBDeviceInterfaceUserClient
deny(1) iokit-open ProvInfoIOKitUserClient // Wow!
deny(1) iokit-open RootDomainUserClient
deny(1) iokit-open com_apple_driver_FairPlayIOKitUserClient
deny(1) iokit-open com_apple_driver_KeyDeliveryIOKitUserClient
deny(1) iokit-open com_apple_driver_KeyDeliveryIOKitUserClientMSE // Wow!
```

CVE-2019-8795

CVE-2019-8794

CVE-2019-7287

Still 0day ?

com_apple_driver_KeyDeliveryIOKitUserClientMSE 0Day

```
KEXT_OBJ:******** 966 *******
(0xfffffff0088041b0)->OSMetaClass:OSMetaClass call 4 args list
x0:0xfffffff00921d7b0
x1:com_apple_driver_KeyDeliveryIOKitUserClientMSE
x2:0xfffffff0091efde8
x3:0xf0
vtable start from addr 0xfffffff007a7d4b8
Inheritance relationship: IOUserClient->IOService->IORegistryEntry->OSObject

override: IOUserClient_destructor1 loc:0xfffffff007a7d4b8 imp:0xfffffff008803d6c
override: IOUserClient_destructor2 loc:0xfffffff007a7d4c0 imp:0xfffffff008803d70
override: IOUserClient_getMetaClass loc:0xfffffff007a7d4f0 imp:0xfffffff008803d88
override: IOService_start loc:0xfffffff007a7d768 imp:0xfffffff008803e7c
override: IOService_stop loc:0xfffffff007a7d770 imp:0xfffffff008803ed0
override: IOUserClient_initWithTask loc:0xfffffff007a7da10 imp:0xfffffff008803e30
override: IOUserClient_clientClose loc:0xfffffff007a7da18 imp:0xfffffff008803ee0
override: IOUserClient_clientDied loc:0xfffffff007a7da20 imp:0xfffffff008803f14
override: IOUserClient_getTargetAndMethodForIndex loc:0xfffffff007a7da68 imp:0xfffffff008803de4
```

- ::clientClose race condition in com_apple_driver_KeyDeliveryIOKitUserClientMSE
- Lead to overwriting of physical memory pages with controlled data!

com_apple_driver_KeyDeliveryIOKitUserClientMSE 0Day

- ::clientClose race condition could apply to all IOKit drivers
  - Setup two threads to race, one is calling ::externalMethod, and the other one is closing the UserClient connection (it triggers ::clientClose)
- It was popular back in Yosemite era, while kernel null-reference still is exploitable
  - I MISS THAT TIME!

com_apple_driver_KeyDeliveryIOKitUserClientMSE 0Day

```
IOReturn __cdecl com_apple_driver_KeyDeliveryIOKitUserClientMSE_clientClose
{
  if ( *(_QWORD *)&this->pad[216] )
    *(_QWORD *)&this->pad[216] = 0LL;              // ->owner_task
  *(_QWORD *)&this->pad[208] = 0LL;
  ((void (*)(void))this->v->IOService_terminate)();
  return 0;
}
```

1. ::ClientClose reset ->owner_task to NULL

```
void __cdecl com_apple_driver_KeyDeliveryIOKitUserClientMSE_sub_FFFFFFF008803F60(
{
  __int64 v2; // x2
  __int64 output_stru_1; // x19
  void **input_stru_1; // x22
  com_apple_driver_KeyDeliveryIOKitUserClientMSE *v5; // x23
  IOMemoryDescriptor *v6; // x0
  IOMemoryDescriptor *v7; // x20
  unsigned int v8; // w26
  IOMemoryMap *v9; // x0
  IOMemoryMap *v10; // x21
  IOMemoryDescriptor *v11; // x0
  IOMemoryDescriptor *v12; // x24
  IOMemoryMap *v13; // x0
  IOMemoryMap *v14; // x25
  __int64 v15; // x26
  __int64 v16; // x0

  output_stru_1 = v2;
  input_stru_1 = (void **)a2;
  v5 = this;
  v6 = IOMemoryDescriptor::withAddressRange(
         *(void **)a2,
         (void *)*((unsigned int *)a2 + 2),
         (void *)2,                                   // kIODirectionOut/Writing
         *(void **)&this->pad[216]);                  // ->owner_task
  if ( !v6 )
    goto LABEL_12;
  v7 = v6;
  v8 = 0xE00002BD;
  v9 = v6->v->IOMemoryDescriptor_map(v6, 0x1000u);
  if ( v9 )
  {
```

2. In one of the external method, it created a memory descriptor instance for memory writing with ->owner_task

if race succeeded, ->owner_task will be NULL

```
static IOMemoryDescriptor * withAddressRange(
    mach_vm_address_t address,
    mach_vm_size_t    length,
    IOOptionBits      options,
    task_t            task);

@function withAddressRanges
 @abstract Create an IOMemoryDescriptor to describe one or more virtual ranges.
 @discussion This method creates and initializes an IOMemoryDescriptor for memory consisting of an array of virtual memory ranges each
specified source task.  This memory descriptor needs to be prepared before it can be used to extract data from the memory described.
@param ranges An array of IOAddressRange structures which specify the virtual ranges in the specified map which make up the memory to
IOAddressRange is the 64bit version of IOVirtualRange.
@param rangeCount The member count of the ranges array.
@param options
    kIOMemoryDirectionMask (options:direction) This nibble indicates the I/O direction to be associated with the descriptor, which ma
    operation of the prepare and complete methods on some architectures.
    kIOMemoryAsReference   For options:type = Virtual or Physical this indicate that the memory descriptor need not copy the ranges a
    local memory.  This is an optimisation to try to minimise unnecessary allocations.
@param task The task each of the virtual ranges are mapped into. Note that unlike IOMemoryDescriptor::withAddress(), kernel_task memo
explicitly prepared when passed to this api. The task argument may be NULL to specify memory by physical address.
@result The created IOMemoryDescriptor on success, to be released by the caller, or zero on failure. */
```

3. task=NULL is to specify memory by physical address

zecOps

Some security highlights about M1 and macOS 11:

- 1. It's difficult to achieve kernel code execution with Kernel PAC that comes with the M1 chip
- 2. Important kernel variables such as csr_config that directly affect CSR/SIP policies are now stored in the read-only segment. Just as kernel code, they are protected by KTRR/CTRR from being modified even after the attacker gain kernel R/W ability. Intel-based Macs do not have this security feature. Read pmap.c and arm_vm_init.c to learn more.
- 3. AuxKC prevents attackers from loading custom kexts immediately after the kernel is exploited. The custom kext gives attackers the ability to deploy an advanced and undetectable payload.
- According to Apple Platform Security PDF. Starting with macOS 11, kext can't be loaded into the kernel on demand without an occurrence of a system reboot. which was not needed in the past.
- 4. APFS snapshot, more steps are needed to modify the root file system.

Thank you

cccccc3742@protonmail.com