Viewer discretion is advised

Decoding an iOS vulnerability

Adam Donenfeld
Zimperium

To read the phrack paper during the presentation:

Agenda

- Sandbox concepts
- Apple user to kernel sandboxing
- New iOS vulnerability
- Tracing the iOS kernel
- Debugging Apple processes
- Zero-clicks, mediaserverd (and PAC ☹)
- Summary

~ $ whoami

ADAM DONENFELD (@doadam)

- Years of experience in research (both PC and mobile)
- Vulnerability assessment
- Vulnerability exploitation
- Senior security researcher at Zimperium
- Presented in world famous security conferences

Sandboxing concepts

- Isolating low-level and high-level processes
  - Narrowed attack surface
  - Preventing leak of sensitive information
- Examples
  - An app is not supposed to have access to biometric information
  - Coreauthd is not supposed to have access to your calendar

Sandboxing concepts

More concrete examples

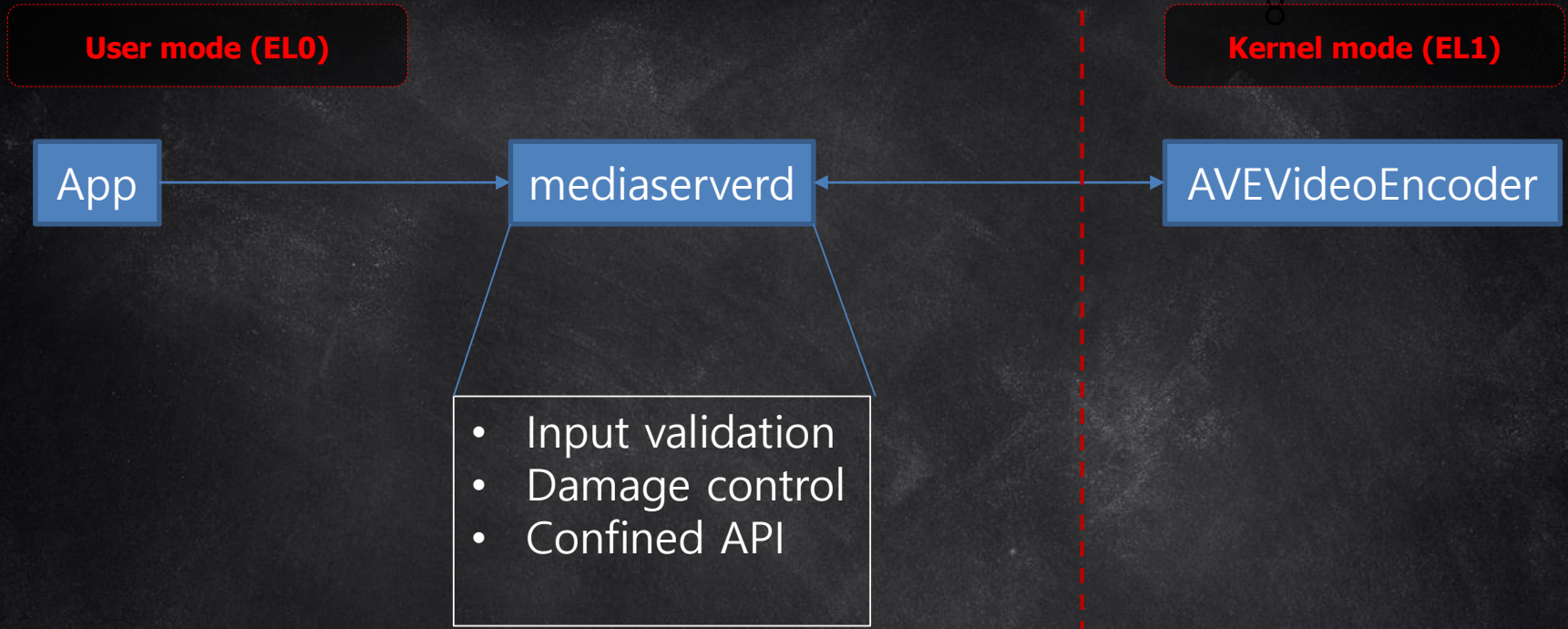
- CVE-2015-7006
  - Airdrop attack – arbitrary file write via sharingd
    - Sharingd is now sandboxed
- CVE-billions-of-them
  - AFC symlinks restrictions
- ZiVA
  - Fully working exploit, still needed sandbox escape

Apple user-to-kernel sandboxing

- Sometimes a feature is required
  - But in a limited manner

- Isolate user and kernel module with a process in between


User mode (EL0)
Kernel mode (EL1)
App
AVEVideoEncoder


User mode (EL0)
Kernel mode (EL1)
App
mediaserverd
AVEVideoEncoder
• Input validation
• Damage control
• Confined API

IOSurface

Helping hackers since 2007

IOSurface < 10.3

Plane check sign mismatch vulnerability

```
check_plane_overlap
    LDURSW          X11, [X10,#-0xC]    ; X11 = plane_base
    LDR             W12, [X10]          ; W12 = plane_size
    ADD             X11, X12, X11       ; Rd = Op1 + Op2
    CMP             X8, X11             ; X8 = last plane base
    B.CC            plane_base_overlaps ; Branch
    ADD             W9, W9, #1          ; Rd = Op1 + Op2
    ADD             X10, X10, #0x50     ; next plane
    CMP             W9, W25; CMP current_plane_id, last_plane_id
    B.CC            check_plane_overlap ; Branch
```

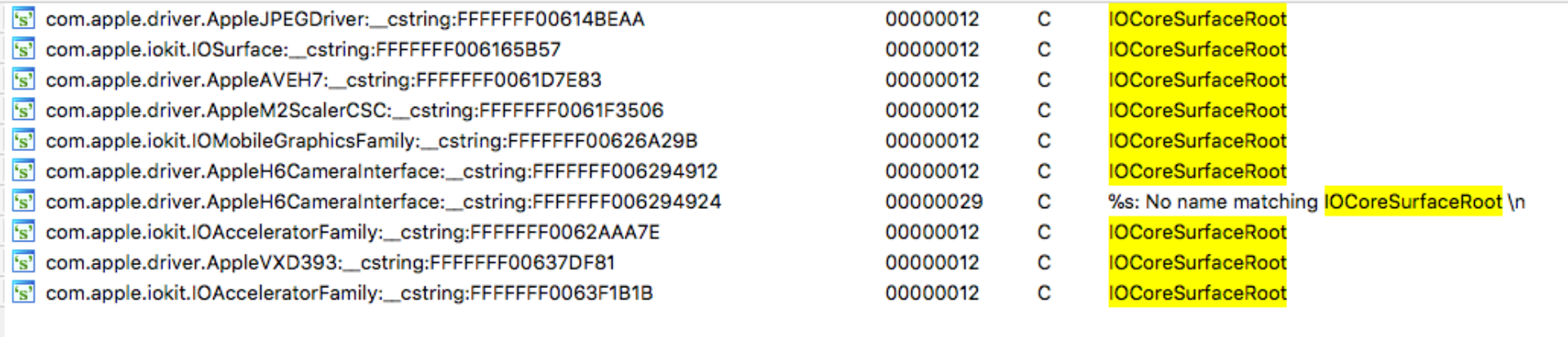
IOSurface

What uses IOSurface?

- IOSurface objects created with *IOSurfaceRootUserClient*
- Looking up IOSurface IDs is done with *IOSurfaceRoot*

- *IOSurfaceRoot* registers itself as “IOCoreSurfaceRoot”

- Lookup IOCoreSurfaceRoot strings



com.apple.driver.AppleJPEGDriver:__cstring:FFFFFFF00614BEAA	00000012	C	IOCoreSurfaceRoot
com.apple.iokit.IOSurface:__cstring:FFFFFFF006165B57	00000012	C	IOCoreSurfaceRoot
com.apple.driver.AppleAVEH7:__cstring:FFFFFFF0061D7E83	00000012	C	IOCoreSurfaceRoot
com.apple.driver.AppleM2ScalerCSC:__cstring:FFFFFFF0061F3506	00000012	C	IOCoreSurfaceRoot
com.apple.iokit.IOMobileGraphicsFamily:__cstring:FFFFFFF00626A29B	00000012	C	IOCoreSurfaceRoot
com.apple.driver.AppleH6CameraInterface:__cstring:FFFFFFF006294912	00000012	C	IOCoreSurfaceRoot
com.apple.driver.AppleH6CameraInterface:__cstring:FFFFFFF006294924	00000029	C	%s: No name matching IOCoreSurfaceRoot \n
com.apple.iokit.IOAcceleratorFamily:__cstring:FFFFFFF0062AAA7E	00000012	C	IOCoreSurfaceRoot
com.apple.driver.AppleVXD393:__cstring:FFFFFFF00637DF81	00000012	C	IOCoreSurfaceRoot
com.apple.iokit.IOAcceleratorFamily:__cstring:FFFFFFF0063F1B1B	00000012	C	IOCoreSurfaceRoot

IOSurface

What uses IOSurface->getPlaneBase\Size() ?

- String lookup, “plane” in those drivers

- “outWidth > pIOSurfaceDst->getPlaneWidth(0) || outHeight > pIOSurfaceDst->getPlaneHeight(0) || outWidth == 0 || outHeight == 0" failed in "/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleD5500/AppleD5500-134.1/AppleD5500.cpp”

IOSurface usage

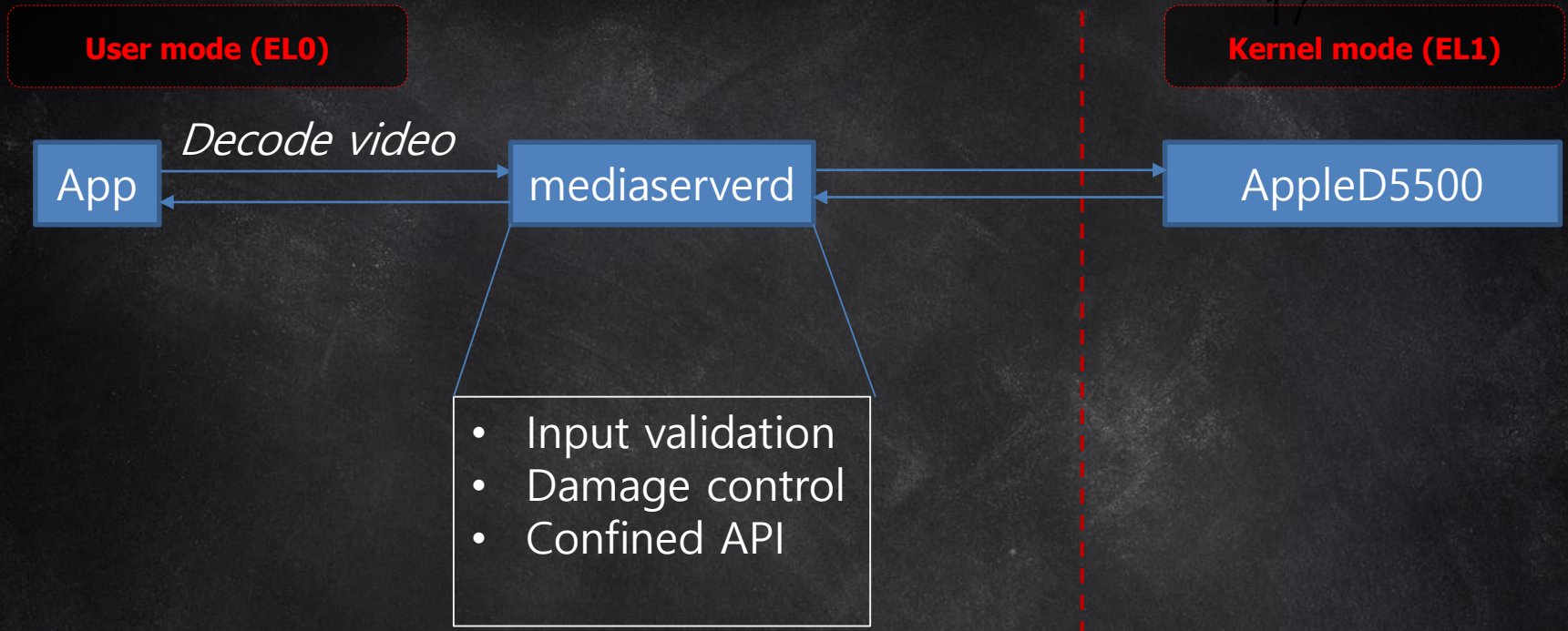
```
    LDR             X8, [X8,#0x110]      ; Load from Memory
    MOV             W1, #1               ; Rd = Op2
    MOV             X0, X19              ; Rd = Op2
    BLR             X8                   ; IOSurface->getPlaneSize(1)
    MOV             X23, X0              ; X23 = second_plane_size
    SXTW            X2, W20              ; Signed Extend Word
    MOV             W1, #0x80            ; Rd = Op2
    MOV             X0, X25              ; Rd = Op2
memset(something, 0x80, first_plane_size)
    BL              memset               ; Branch with Link
    SXTW            X2, W23              ; Signed Extend Word
    MOV             W1, #0x80            ; Rd = Op2
    MOV             X0, X27              ; Rd = Op2
memset(something, 0x80, second_plane_size)
    BL              memset               ; Branch with Link
```

New primitive!

- Override something with 0x80s, arbitrary length

Who are you?

- Video-decoding driver
- Closed from sandbox
- Communication is done via mediaserverd


User mode (EL0)
Kernel mode (EL1)
Decode video
App
mediaserverd
AppleD5500
• Input validation
• Damage control
• Confined API

IOSurface < 10.3

Plane check sign mismatch vulnerability

```
check_plane_overlap
    LDURSW          X11, [X10,#-0xC]     ; X11 = plane_base
    LDR             W12, [X10]           ; W12 = plane_size
    ADD             X11, X12, X11        ; Rd = Op1 + Op2
    CMP             X8, X11              ; X8 = last plane base
    B.CC            plane_base_overlaps  ; Branch
    ADD             W9, W9, #1           ; Rd = Op1 + Op2
    ADD             X10, X10, #0x50      ; next plane
    CMP             W9, W25; CMP current_plane_id, last_plane_id
    B.CC            check_plane_overlap  ; Branch
```

```
check_plane_overlap
    LDUR            X11, [X10,#-0xC]     ; X11 = plane_base
    LDR             W12, [X10]           ; W12 = plane_size
    ADD             X11, X12, X11        ; Rd = Op1 + Op2
    CMP             X8, X11              ; X8 = last plane base
    B.CC            plane_base_overlaps  ; Branch
    ADD             W9, W9, #1           ; Rd = Op1 + Op2
    ADD             X10, X10, #0x50      ; next plane
    CMP             W9, W25; CMP current_plane_id, last_plane_id
    B.CC            check_plane_overlap  ; Branch
```

IOSurface usage

```
    LDR             X8, [X8,#0x110]      ; Load from Memory
    MOV             W1, #1               ; Rd = Op2
    MOV             X0, X19              ; Rd = Op2
    BLR             X8                   ; IOSurface->getPlaneSize(1)
    MOV             X23, X0              ; X23 = second_plane_size
    SXTW            X2, W20              ; Signed Extend Word
    MOV             W1, #0x80            ; Rd = Op2
    MOV             X0, X25              ; Rd = Op2
memset(something, 0x80, first_plane_size)
    BL              memset               ; Branch with Link
    SXTW            X2, W23              ; Signed Extend Word
    MOV             W1, #0x80            ; Rd = Op2
    MOV             X0, X27              ; Rd = Op2
memset(something, 0x80, second_plane_size)
    BL              memset               ; Branch with Link
```

How is the IOSurface ID supplied?

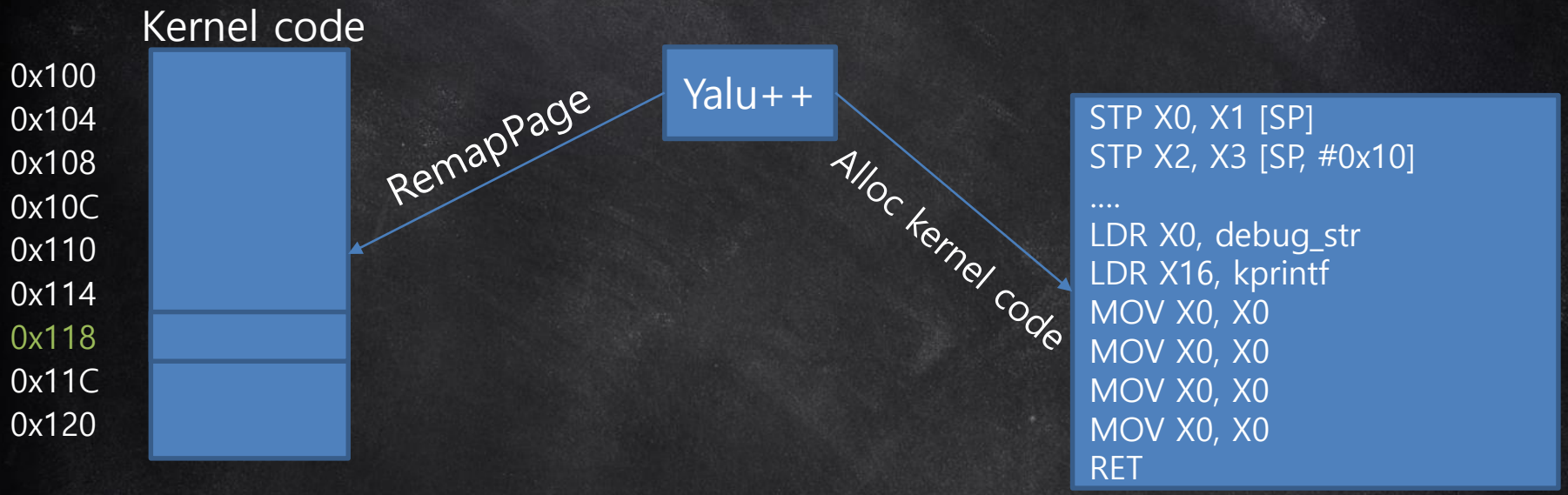
- Look up xrefs to this function...
  - Leads to a virtual function
- ~20 functions from entry point to the externalMethod
- How to make sure the function is reached?

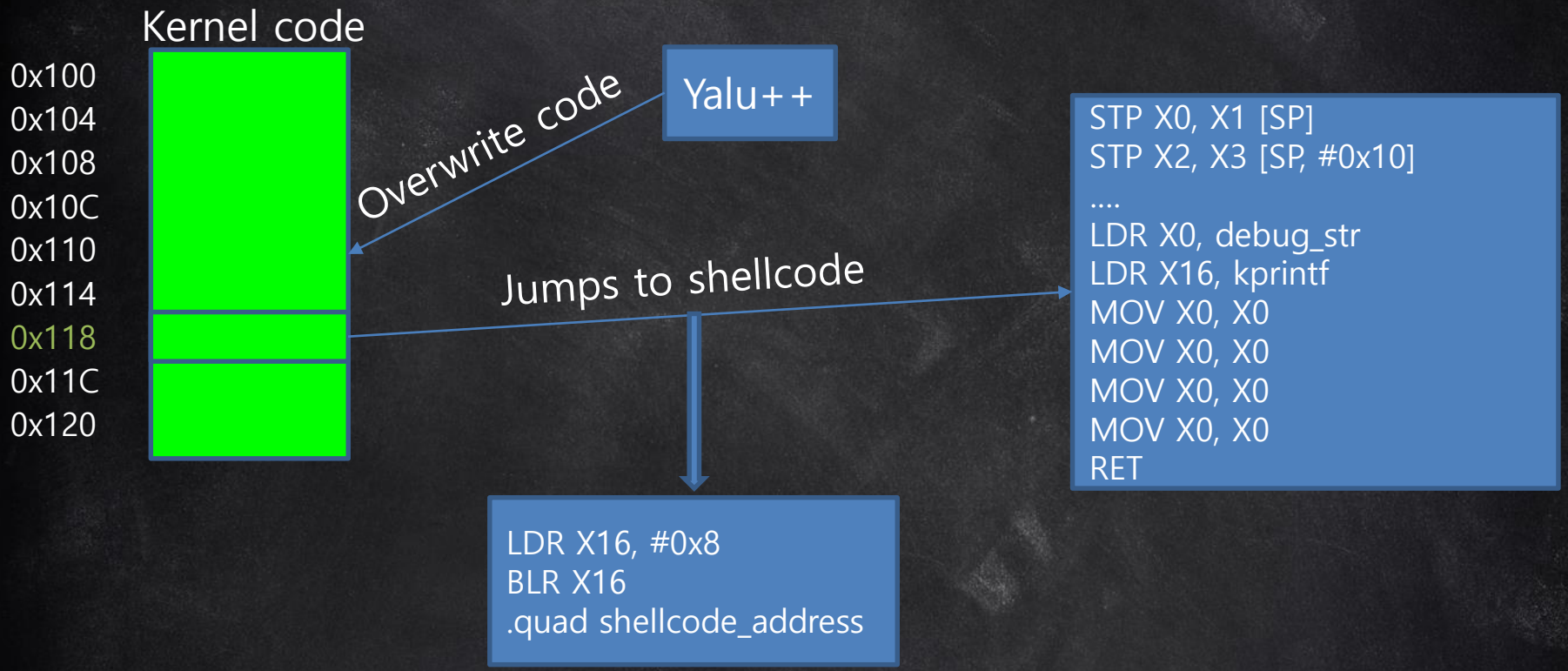
KPP bypass

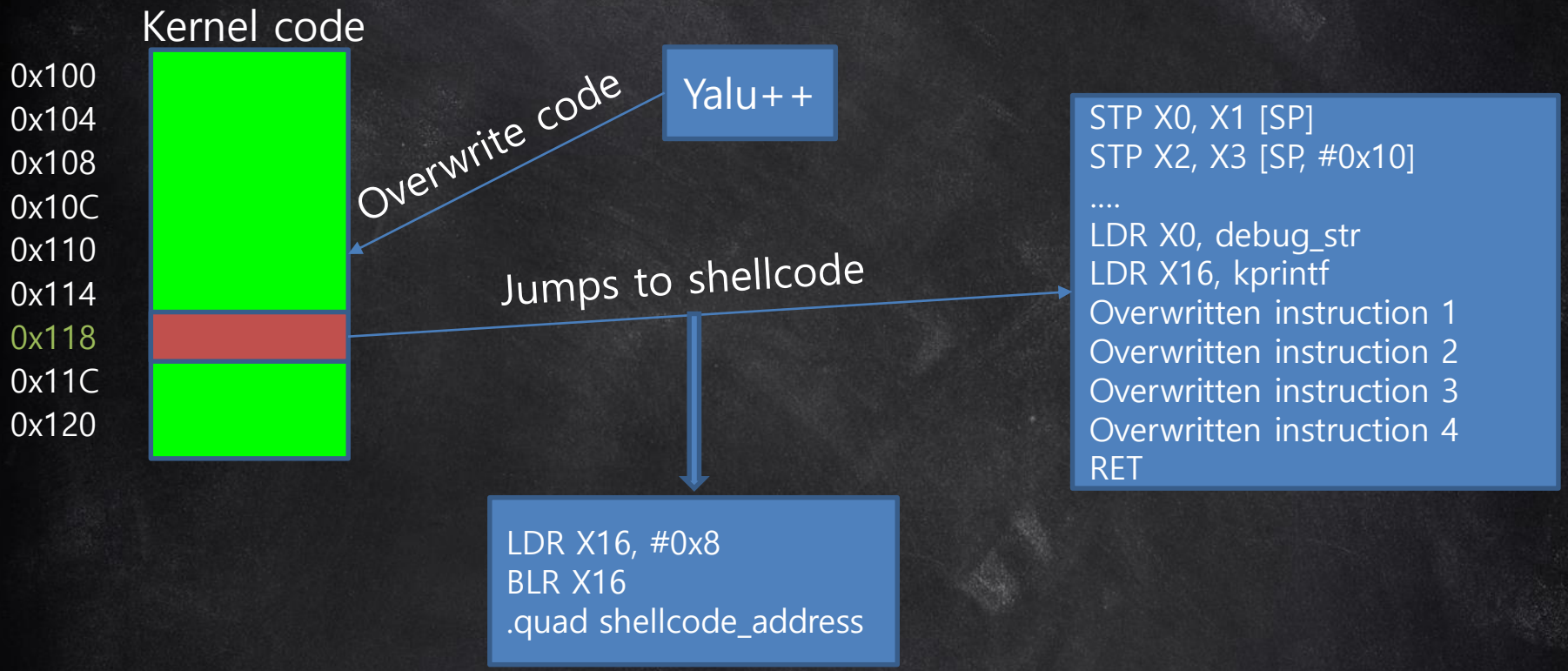
"Hook" kernel functions

- Modified Yalu* project
- "RemapPage" where we want to hook
  - Disables KPP for this certain page
- Overwrite 4 instructions:
  - *LDR X16, #0x8*
  - *BLR X16*
  - *8 bytes address of shellcode*

- *Special thanks to @qwertyoruiop and @marcograss for their work on that project


Kernel code
0x100
0x104
0x108
0x10C
0x110
0x114
0x118
0x11C
0x120
RemapPage
Yalu++
Alloc kernel code
STP X0, X1 [SP]
STP X2, X3 [SP, #0x10]
....
LDR X0, debug_str
LDR X16, kprintf
MOV X0, X0
MOV X0, X0
MOV X0, X0
MOV X0, X0
RET


Kernel code
0x100
0x104
0x108
0x10C
0x110
0x114
0x118
0x11C
0x120
Overwrite code
Yalu++
Jumps to shellcode
STP X0, X1 [SP]
STP X2, X3 [SP, #0x10]
....
LDR X0, debug_str
LDR X16, kprintf
MOV X0, X0
MOV X0, X0
MOV X0, X0
MOV X0, X0
RET
LDR X16, #0x8
BLR X16
.quad shellcode_address


Kernel code
0x100
0x104
0x108
0x10C
0x110
0x114
0x118
0x11C
0x120
Overwrite code
Yalu++
Jumps to shellcode
STP X0, X1 [SP]
STP X2, X3 [SP, #0x10]
....
LDR X0, debug_str
LDR X16, kprintf
Overwritten instruction 1
Overwritten instruction 2
Overwritten instruction 3
Overwritten instruction 4
RET
LDR X16, #0x8
BLR X16
.quad shellcode_address

```
LDR             W8, [X19,#0x4E0]     ; Load from Memory
CBNZ            W8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Non-Zero
LDR             X8, [X19,#0x448]     ; Load from Memory
LDRB            W8, [X8,#6]          ; Load from Memory
AND             W8, W8, #0x30        ; Rd = Op1 & Op2
CBNZ            W8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Non-Zero
LDR             X8, [X9,#0x10]       ; Load from Memory
CBZ             X8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Zero
LDR             X10, [X9,#0x40]      ; Load from Memory
ADD             X0, X8, W10,UXTW     ; Rd = Op1 + Op2
LDR             W8, [X9,#0x54]       ; Load from Memory
LSR             W8, W8, #1           ; Logical Shift Right
LSR             X9, X10, #0x20       ; Logical Shift Right
MADD            W2, W8, W9, WZR      ; Multiply-Add
MOV             W1, #0x80            ; Rd = Op2
BL              memset               ; Branch with Link
```

Reversing AppleD5500

IOKit reversing

- Realizing where are the vtables and functions
- Entry points (IOUserClient->externalMethod)
- Should be automated, but can be done manually

Reversing AppleD5500

IOKit reversing

- 0x80 had something to do with IOSurface…
- Let’s examine IOSurfaces in the driver!

- A quick strings search reveals:
  - *“AppleVXD393::allocateKernelMemory kAllocMapTypeIOSurface - loo kupSurface failed. %d \n”*

```
surf_props->plane_offset[0] = v24->vtable->IOSurface_FFFFFFF00668FDD8L)(v24,0LL);
surf_props->plane_offset[1] = v24->vtable->IOSurface_FFFFFFF00668FDD8L)(v24,1LL);
surf_props->plane_bytes_per_row[0] = v24->vtable->IOSurface_FFFFFFF00668FF40L)(v24,0LL);
surf_props->plane_height[0] = v24->vtable->IOSurface_FFFFFFF00668FE8CL)(v24,0LL);
surf_props->plane_height[1] = v24->vtable->IOSurface_FFFFFFF00668FE8CL)(v24,1LL);
surf_props->plane_width[0] = v24->vtable->IOSurface_FFFFFFF00668FE50L)(v24,0LL);
surf_props->plane_width[1] = v24->vtable->IOSurface_FFFFFFF00668FE50L)(v24,1LL);
surf_props->plane_offset_again?[0] = v24->vtable->IOSurface_FFFFFFF00668FDD8L)(v24,0LL);
surf_props->plane_offset_again?[1] = v24->vtable->IOSurface_FFFFFFF00668FDD8L)(v24,1LL);
v31 = surface_descriptor->vtable->__ZN18IOMemoryDescriptor3mapEj((IOMemoryDescriptor *)v17, 0);
if ( v31 )
{
    surf_props->surface_buffer_mapping = v31->vtable->__ZN11IOMemoryMap17getVirtualAd-
dressEv)();
```

Reversing AppleD5500

IOKit reversing

- IOSurface loading code
- Same offsets as used in the memset call
- Kernel tracing technique reveals this is indeed an IOSurface object!

AppleD5500

Who supplies this IOSurface object?

- Mediaserverd calls AppleD5500
  - VideoToolBox, to be accurate
- Let’s reverse VideoToolBox!
  - Contained in dyld_shared_cache
- No IDA 7 back then

VideoToolBox

- No apparent usage of AppleD5500

- Maybe another framework?

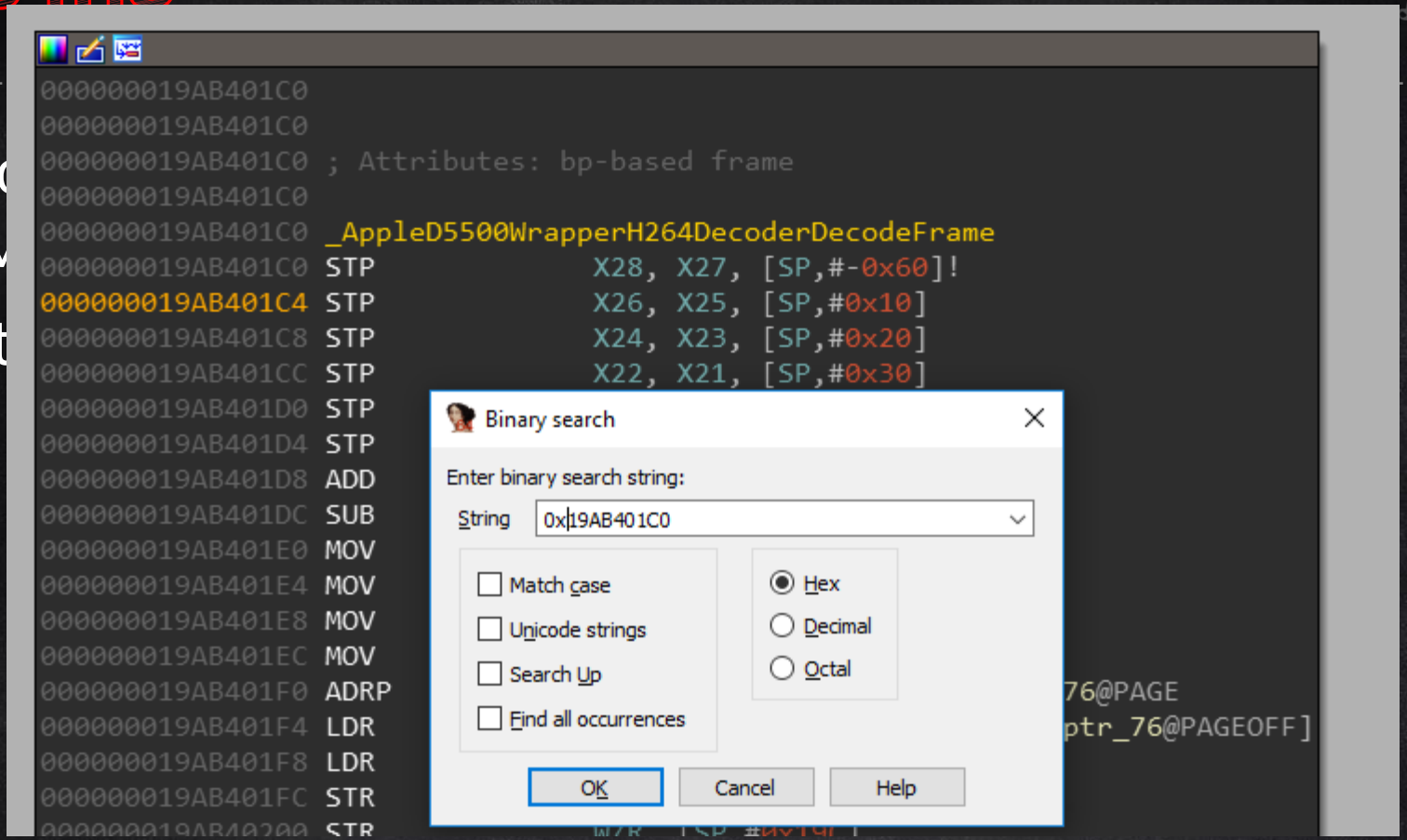
VideoToolBox

- String lookup
- H264H8

- AppleD5500's IOKit external methods are interesting
  - _AppleD5500DecodeFrameInternal
    - IOConnectCallStructMethod
- _AppleD5500WrapperH264DecoderDecodeFrame
  - No xrefs…

H264H8

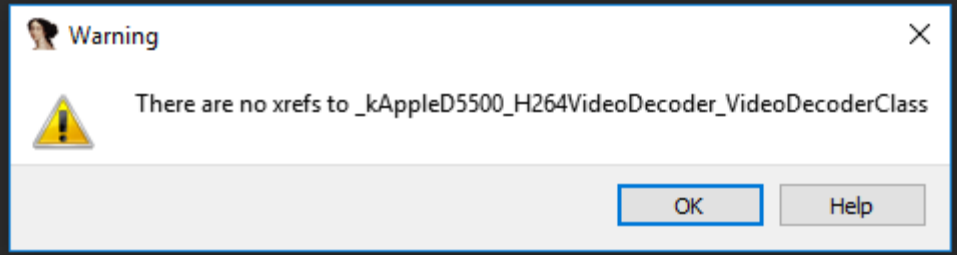
- No
- (M
- Vt



H264H8

-
-

```
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
                DCB     0
_kAppleD5500_H264VideoDecoder_VideoDecoderClass DCB     1
                DCB     0
                DCB     0
                DCB     0
                DCQ _AppleD5500WrapperH264DecoderStartSession
                DCQ _AppleD5500WrapperH264DecoderDecodeFrame
                DCQ _AppleD5500WrapperH264DecoderCopySupportedPropertyDictionary
                DCB     0
                DCB     0
```



```
                ; ORG 0x1A55BA658
_kH264VideoDecoderVTable DCB    0          ; DATA XREF: _AppleD5500WrapperH26
                                           ; _AppleD5500WrapperH264DecoderCre
                DCB    0
                DCB    0
                DCB    0
                DCB    0
                DCB    0
                DCB    0
                DCB    0
                DCQ _kAppleD5500_H264VideoDecoder_BaseClass
                DCQ _kAppleD5500_H264VideoDecoder_VideoDecoderClass
```

```
EXPORT _H264H8Register
_H264H8Register

var_10= -0x10
var_s0=  0

STP             X20, X19, [SP,#-0x10+var_10]!
STP             X29, X30, [SP,#0x10+var_s0]
ADD             X29, SP, #0x10
BL              _AppleD5500CheckPlatform
CMP             W0, #1
B.NE            no_registration
ADRP            X19, #_AppleD5500WrapperH264DecoderCreateInstance@PAGE
ADD             X19, X19, #_AppleD5500WrapperH264DecoderCreateInstance@PAGEOFF
MOV             W0, #0x61760000
MOVK            W0, #0x6331             ; avc1
MOV             X1, X19
BL              j__VTRegisterVideoDecoder_2
MOV             W0, #0x64720000
MOVK            W0, #0x6D69             ; drmi
MOV             X1, X19
BL              j__VTRegisterVideoDecoder_2
MOV             W0, #0x65610000
MOVK            W0, #0x7663             ; eavc
MOV             X1, X19
BL              j__VTRegisterVideoDecoder_2
```

H264H8

- H264H8Register initializes the connection with the driver

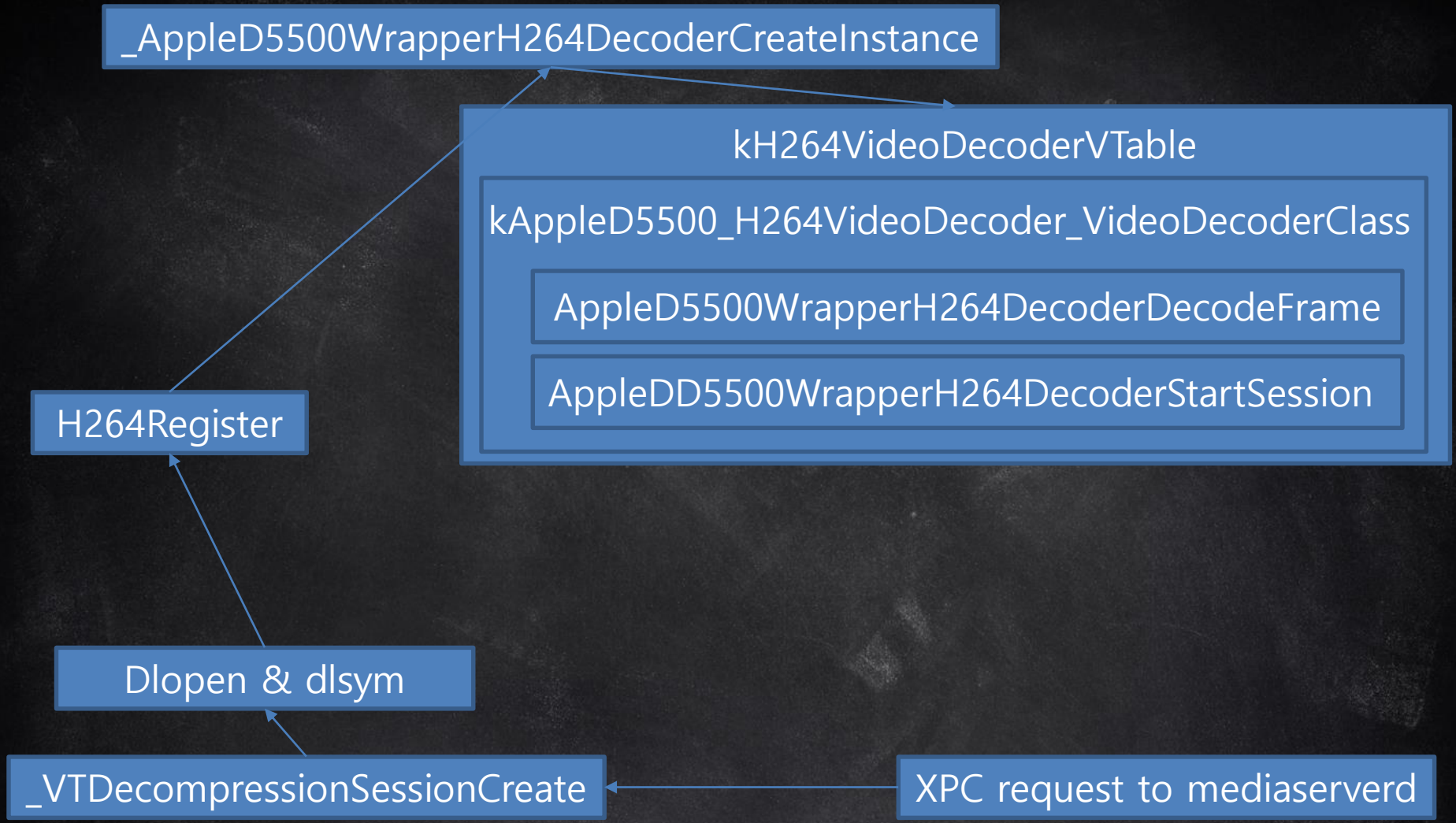
```
ADRP            X0, #aSystemLibraryV_8@PAGE ; "/System/Library/VideoDecoders/H264H8.vi"...
ADD             X0, X0, #aSystemLibraryV_8@PAGEOFF ; "/System/Library/VideoDecoders/H264H8.vi"...
ADRP            X1, #aH264h8register@PAGE ; "H264H8Register"
ADD             X1, X1, #aH264h8register@PAGEOFF ; "H264H8Register"
BL              _VTLoadVideoDecoder
```

```
_VTLoadVideoDecoder

var_10= -0x10
var_s0=  0

STP             X20, X19, [SP,#-0x10+var_10]!
STP             X29, X30, [SP,#0x10+var_s0]
ADD             X29, SP, #0x10
MOV             X19, X1
MOV             W1, #4                  ; mode
BL              j__dlopen_35
CBZ             X0, loc_1841D2FD8
MOV             X1, X19                 ; symbol
BL              _dlsym_0
CBZ             X0, loc_1841D2FD8
LDP             X29, X30, [SP,#0x10+var_s0]
LDP             X20, X19, [SP+0x10+var_10],#0x20
BR              X0
; ---------------------------------------------------------------------------

loc_1841D2FD8                           ; CODE XREF: _VTLoadVideoDecoder+18?j
                                        ; _VTLoadVideoDecoder+24?j
LDP             X29, X30, [SP,#0x10+var_s0]
LDP             X20, X19, [SP+0x10+var_10],#0x20
RET
```


_AppleD5500WrapperH264DecoderCreateInstance
kH264VideoDecoderVTable
kAppleD5500_H264VideoDecoder_VideoDecoderClass
AppleD5500WrapperH264DecoderDecodeFrame
AppleDD5500WrapperH264DecoderStartSession
H264Register
Dlopen & dlsym
_VTDecompressionSessionCreate
XPC request to mediaserverd

VTDecompressionSession

- Documented API
- VTDecompressionSessionDecodeFrame

- Perhaps that’s what initializes the IOSurface object in the kernel?

VTDecompressionSessionDecodeFrame

- Calls to AppleD5500WrapperH264DecoderDecodeFrame
  - Within the H264H8 framework

- And then…

```
tile_decode_dictionary = j__CMGetAttachment_3(v7, CFSTR("tileDecode"), 0LL);
tile_decode_dictionary_1 = tile_decode_dictionary;
if ( tile_decode_dictionary )
{
    canvas_surface_ID1 = 0LL;
    v39 = CFDictionaryGetValue_24(tile_decode_dictionary, CFSTR("canvasSurfaceID"));
    v40 = CFNumberGetValue_22(v39, 10LL, &canvas_surface_ID1);
    ...
    if ( !v40 )
    {
    ...
    ...
    v55 = CFDictionaryGetValue_24(tile_decode_dictionary_1, CFSTR("offsetX"));
    CFNumberGetValue_22(v55, 3LL, &v92);
    v56 = CFDictionaryGetValue_24(tile_decode_dictionary_1, CFSTR("offsetY"));
    CFNumberGetValue_22(v56, 3LL, &v91);
    v57 = CFDictionaryGetValue_24(tile_decode_dictionary_1, CFSTR("lastTile"));
...
```

VTDecompressionSessionDecodeFrame

- Optionally receives an IOSurface ID! (not documented)
- Receives also X and Y offsets…
- Is this the surface in the kernel?
- iOS kernel tracing to the rescue!

AppleD5500 & mediaserverd

Quick recap

- Objective – get to memset
- iOS kernel tracing

```
if ( context->tile_decode )
  {
    dest_surf->tile_decode = 1;
    tile_offset_x = context->tile_offset_x;
    dest_surf->tile_offset_x = tile_offset_x;
    tile_offset_y = context->tile_offset_y;
    dest_surf->tile_offset_y = tile_offset_y;
    v73 = tile_offset_x + tile_offset_y * dest_surf->surf_props.plane_bytes_per_row[0];
    v74 = tile_offset_x
        + ((dest_surf->surf_props.plane_bytes_per_row[1] * tile_offset_y + 1) >> 1)
        + dest_surf->surf_props.plane_offset_again?[1];
    dest_surf->surf_props.plane_offset[0] = v73 + dest_surf->surf_props.plane_offset_again?[0];
    dest_surf->surf_props.plane_offset[1] = v74;
  }
...
   if ( !context->field_4E0 && !(context->some_unknown_data->unk & 0x30) )
    {
      surface_buffer_mapping = v85->surf_props.surface_buffer_mapping;
      if ( surface_buffer_mapping )
        memset_stub(
          (char *)surface_buffer_mapping + (unsigned int)*(_QWORD *)&v85->surf_props.plane_offset[1],
          0x80LL,
   ((dest_surf->surf_props.plane_height[0] >> 1) *
(*(_QWORD *)&dest_surf->surf_props.plane_offset[1] >> 0x20)));
    }
```

AppleD5500

- Closed source driver
- No xrefs
- How to find out what that field is?

```
LDR             W8, [X19,#0x4E0]     ; Load from Memory
CBNZ            W8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Non-Zero
LDR             X8, [X19,#0x448]     ; Load from Memory
LDRB            W8, [X8,#6]          ; Load from Memory
AND             W8, W8, #0x30        ; Rd = Op1 & Op2
CBNZ            W8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Non-Zero
LDR             X8, [X9,#0x10]       ; Load from Memory
CBZ             X8, loc_FFFFFFF006C4E7DC ; Compare and Branch on Zero
LDR             X10, [X9,#0x40]      ; Load from Memory
ADD             X0, X8, W10,UXTW     ; Rd = Op1 + Op2
LDR             W8, [X9,#0x54]       ; Load from Memory
LSR             W8, W8, #1           ; Logical Shift Right
LSR             X9, X10, #0x20       ; Logical Shift Right
MADD            W2, W8, W9, WZR      ; Multiply-Add
MOV             W1, #0x80            ; Rd = Op2
BL              _memset.stub         ; Branch with Link
```

AppleD5500

- Closed source driver
- No xrefs
- How to find out what that field is?

- Dump the entire driver’s text section and grep

```
Adam-MBP16:tmp adam$ cat d5500 | grep 448 | grep STR
0xffffffff006c30448L      STR             D1, [X19,#0xA90]; Store to Memory
0xffffffff006c41448L      STRB            W13, [X1]; Store to Memory
0xffffffff006c44488L      STRH            W17, [X13,X15,LSL#1]; Store to Memory
0xffffffff006c44810L      STR             W8, [SP,#0x90+var_54]; Store to Memory
0xffffffff006c4481cL      STR             W8, [X19,#0x64C]; Store to Memory
0xffffffff006c4483cL      STR             X8, [SP,#0x90+var_88]; Store to Memory
0xffffffff006c44848L      STR             X8, [SP,#0x90+var_90]; Store to Memory
0xffffffff006c44860L      STR             X8, [SP,#0x90+var_88]; Store to Memory
0xffffffff006c4486cL      STR             X8, [SP,#0x90+var_90]; Store to Memory
0xffffffff006c44890L      STRB            W9, [X8,#6]; Store to Memory
0xffffffff006c4489cL      STR             X8, [SP,#0x90+var_88]; Store to Memory
0xffffffff006c448a8L      STR             X8, [SP,#0x90+var_90]; Store to Memory
0xffffffff006c448c0L      STR             X8, [SP,#0x90+var_88]; Store to Memory
0xffffffff006c448ccL      STR             X8, [SP,#0x90+var_90]; Store to Memory
0xffffffff006c448e8L      STR             W9, [X8,#4]; Store to Memory
0xffffffff006c448f4L      STR             X8, [SP,#0x90+var_88]; Store to Memory
0xffffffff006c47448L      STRB            W0, [X19,#0x2A0]; Store to Memory
0xffffffff006c495ccL      STR             X9, [X10,#0x448]; Store to Memory
0xffffffff006c50448L      STR             W24, [X22,#0x17BC]; Store to Memory
Adam-MBP16:tmp adam$
```

```
FFFFFFF006C49594 var_40= -0x40
FFFFFFF006C49594 var_30= -0x30
FFFFFFF006C49594 var_20= -0x20
FFFFFFF006C49594 var_10= -0x10
FFFFFFF006C49594 var_s0=  0
FFFFFFF006C49594
FFFFFFF006C49594 STP             X26, X25, [SP,#-0x10+var_40]! ; Store Pair
FFFFFFF006C49598 STP             X24, X23, [SP,#0x40+var_30] ; Store Pair
FFFFFFF006C4959C STP             X22, X21, [SP,#0x40+var_20] ; Store Pair
FFFFFFF006C495A0 STP             X20, X19, [SP,#0x40+var_10] ; Store Pair
FFFFFFF006C495A4 STP             X29, X30, [SP,#0x40+var_s0] ; Store Pair
FFFFFFF006C495A8 ADD             X29, SP, #0x40              ; Rd = Op1 + Op2
FFFFFFF006C495AC MOV             X20, X4                    ; Rd = Op2
FFFFFFF006C495B0 MOV             X21, X2                    ; Rd = Op2
FFFFFFF006C495B4 MOV             X22, X1                    ; Rd = Op2
FFFFFFF006C495B8 MOV             X19, X0                    ; Rd = Op2
FFFFFFF006C495BC MOV             X8, #0                     ; Rd = Op2
FFFFFFF006C495C0 LDR             X24, [X19,#0x1C8]          ; Load from Memory
FFFFFFF006C495C4 LDR             X9, [X19,#0x28]            ; Load from Memory
FFFFFFF006C495C8 LDR             X10, [X19,#0x30F8]         ; Load from Memory
FFFFFFF006C495CC STR             X9, [X10,#0x448]           ; Store to Memory
```

```
LDR     X11, [X19,#0x1B0]     ; Load from Memory
LDRH    W11, [X11,#0x24]      ; Load from Memory
LDR     X12, [X19,#0x28]      ; Load from Memory
LDRH    W13, [X12,#6]         ; Load from Memory
MOV     W14, #0xFFCF          ; Rd = Op2
AND     W13, W13, W14         ; Rd = Op1 & Op2
BFI     W13, W11, #4, #2      ; Bit Field Insert
STRH    W13, [X12,#6]         ; Store to Memory
```

W13 = (W11 & 3) * 0x10

```
LDR     X11, [X19,#0x1B0]     ; Load from Memory
LDRH    W11, [X11,#0x24]      ; Load from Memory
LDR     X12, [X19,#0x28]      ; Load from Memory
LDRH    W13, [X12,#6]         ; Load from Memory
MOV     W14, #0xFFCF          ; Rd = Op2
AND     W13, W13, W14         ; Rd = Op1 & Op2
BFI     W13, W11, #4, #2      ; Bit Field Insert
STRH    W13, [X12,#6]         ; Store to Memory
```

```
Adam-MBP16:tmp adam$ cat d5500 | grep 0x1B0 | grep STR
0xfffffff006c43f14L     STR             X8, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f48L     STR             X8, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f54L     STR             XZR, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f60L     STR             XZR, [X19,#0x1B0]; Store to Memory
0xfffffff006c46108L     STRB            WZR, [X19,#0x1B0]; Store to Memory
Adam-MBP16:tmp adam$
```

CH264Decoder::DecodeStream error h264fw_SetPpsAndSps

~ $ whoami

ADAM DONENFELD (@doadam)

- Years of experience in research (both PC and mobile)
- Vulnerability assessment
- Vulnerability exploitation
- Senior security researcher at Zimperium
- Presented in world famous security conferences

~ $ whoami

ADAM DONENFELD (@doadam)

- Years of experience in research (both PC and mobile)
- Vulnerability assessment
- Vulnerability exploitation
- Senior security researcher at Zimperium
- Presented in world famous security conferences
- Never really liked H264

AppleD5500

- Further looking up the source of the check…
- Arriving at a function with the following string:
- AVC_Decoder::ParseHeader unsupported naluLengthSize

- Googling “AVC nalu”
- First result is “Introduction to H.264: (1) NAL Unit”

- H.264 standard http://www.itu.int/rec/T-REC-H.264/e

H.264 format

H.264 in 60 seconds

- A packed video consists of “NAL units”

7.3.1 NAL unit syntax

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
NumBytesInRBSP = 0		
nalUnitHeaderBytes = 1		
if(nal_unit_type == 14 || nal_unit_type == 20 || nal_unit_type == 21) {		
if(nal_unit_type != 21)		
svc_extension_flag	All	u(1)
else		
avc_3d_extension_flag	All	u(1)
if(svc_extension_flag) {		
nal_unit_header_svc_extension( ) /* specified in Annex G */	All	
nalUnitHeaderBytes += 3		
} else if(avc_3d_extension_flag) {		
nal_unit_header_3davc_extension( ) /* specified in Annex J */		
nalUnitHeaderBytes += 2		
} else {		
nal_unit_header_mvc_extension( ) /* specified in Annex H */	All	
nalUnitHeaderBytes += 3		
}		
}		
for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
}		
}		

H.264 format

H.264 in 60 seconds

- A packed video consists of “NAL units”
- Each NAL unit has a type
- The NAL unit is built according to its type
- How to find its type?

```
LDP             W9, W8, [X19,#0x18]     ; Load Pair
CBNZ            W9, parse_nal_by_type   ; Compare and Branch on Non-Zero
CMP             W8, #5                  ; Set cond. codes on Op1 - Op2
B.EQ            idr_type_and_no_idc_ref ; Branch

parse_nal_by_type
SUB             W9, W8, #1              ; switch 12 cases
CMP             W9, #0xB                ; Set cond. codes on Op1 - Op2
B.HI            def_FFFFFFF006C3A2DC    ; jumptable FFFFFFF006C3A2DC default case
ADRP            X10, #jpt_FFFFFFF006C3A2DC@PAGE ; Address of Page
ADD             X10, X10, #jpt_FFFFFFF006C3A2DC@PAGEOFF ; Rd = Op1 + Op2
LDRSW           X9, [X10,X9,LSL#2]      ; Load from Memory
ADD             X9, X9, X10             ; Rd = Op1 + Op2
BR              X9                      ; switch jump

idr_type_and_no_idc_ref
ADRP            X0, #aZeroNal_ref_id@PAGE ; "zero nal_ref_idc with IDR!"
ADD             X0, X0, #aZeroNal_ref_id@PAGEOFF ; "zero nal_ref_idc with IDR!"
BL              kprintf                 ; Branch with Link
MOV             W0, #0x131              ; Rd = Op2
B               cleanup                 ; Branch
```

Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C	Annex A NAL unit type class	Annex G and Annex H NAL unit type class	Annex I and Annex J NAL unit type class
0	Unspecified		non-VCL	non-VCL	non-VCL
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4	VCL	VCL	VCL
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2	VCL	not applicable	not applicable
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3	VCL	not applicable	not applicable
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4	VCL	not applicable	not applicable
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3	VCL	VCL	VCL
6	Supplemental enhancement information (SEI) sei_rbsp( )	5	non-VCL	non-VCL	non-VCL
7	Sequence parameter set seq_parameter_set_rbsp( )	0	non-VCL	non-VCL	non-VCL
8	Picture parameter set pic_parameter_set_rbsp( )	1	non-VCL	non-VCL	non-VCL
9	Access unit delimiter access_unit_delimiter_rbsp( )	6	non-VCL	non-VCL	non-VCL
10	End of sequence end_of_seq_rbsp( )	7	non-VCL	non-VCL	non-VCL
11	End of stream end_of_stream_rbsp( )	8	non-VCL	non-VCL	non-VCL
12	Filler data filler_data_rbsp( )	9	non-VCL	non-VCL	non-VCL
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	10	non-VCL	non-VCL	non-VCL
14	Prefix NAL unit prefix_nal_unit_rbsp( )	2	non-VCL	suffix dependent	suffix dependent
15	Subset sequence parameter set subset_seq_parameter_set_rbsp( )	0	non-VCL	non-VCL	non-VCL
16	Depth parameter set depth_parameter_set_rbsp( )	11	non-VCL	non-VCL	non-VCL
17..18	Reserved		non-VCL	non-VCL	non-VCL
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	2, 3, 4	non-VCL	non-VCL	non-VCL
20	Coded slice extension slice_layer_extension_rbsp( )	2, 3, 4	non-VCL	VCL	VCL
21	Coded slice extension for a depth view component or a 3D-AVC texture view component slice_layer_extension_rbsp( )	2, 3, 4	non-VCL	non-VCL	VCL
22..23	Reserved		non-VCL	non-VCL	VCL
24..31	Unspecified		non-VCL	non-VCL	non-VCL

H.264 format

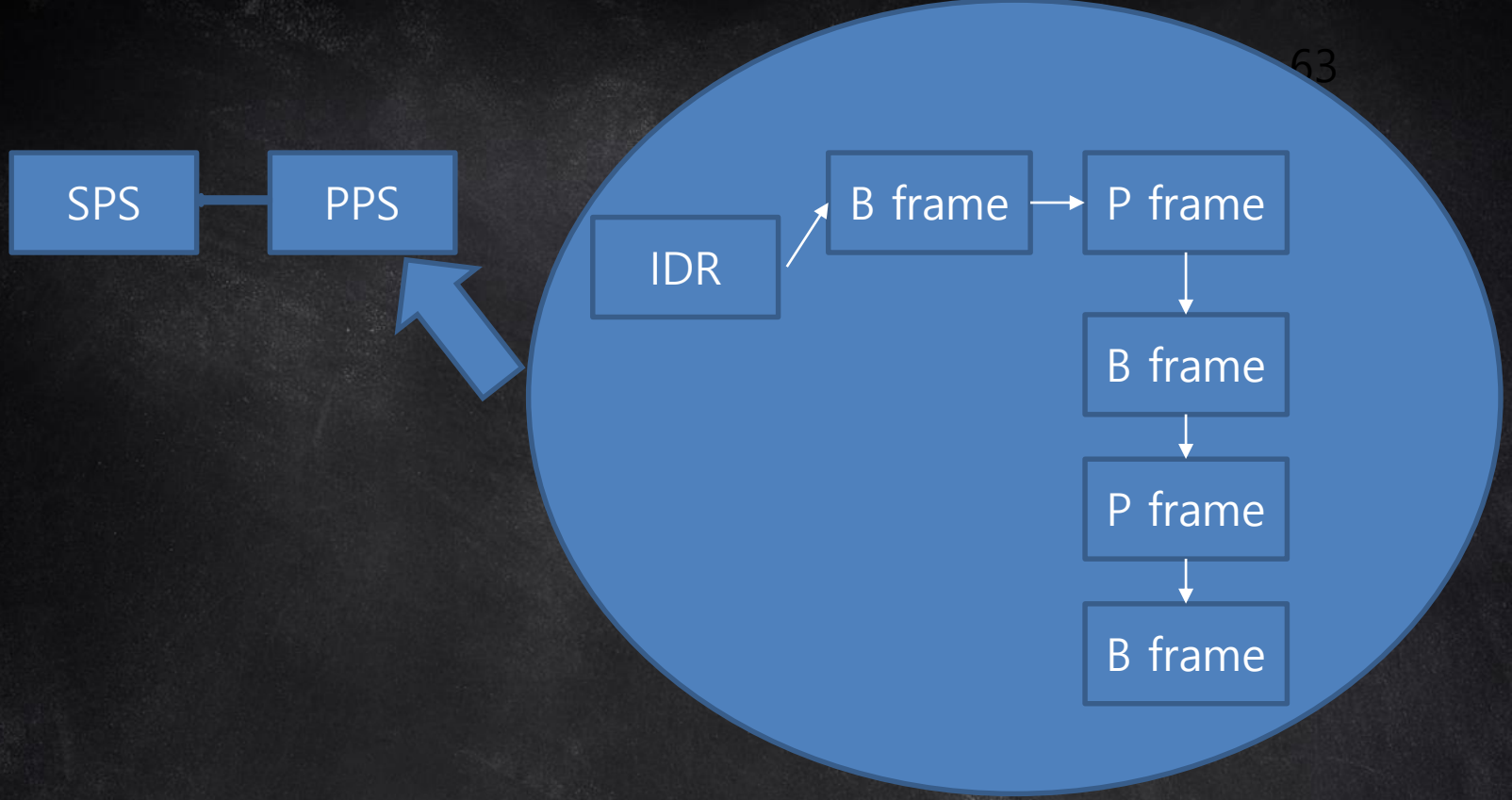
H.264 in 60 seconds

- A packed video consists of “NAL units”
- Each NAL unit has a type
- The NAL unit is built according to its type
- Each type is parsed separatedly

H.264 format

H.264 in 60 seconds

- NAL types
- SPS (sequence parameter set)
  - General properties for coded video sequence
- PPS (picture parameter set)
  - General properties for coded picture sequence
- IDR (Instantaneous Decoding Refresh)
  - First NAL in a coded video sequence
    - For each SPS, the first NAL is an IDR NAL


SPS
PPS
IDR
B frame
P frame
B frame
P frame
B frame

```
LDR     X11, [X19,#0x1B0]     ; Load from Memory
LDRH    W11, [X11,#0x24]      ; Load from Memory
LDR     X12, [X19,#0x28]      ; Load from Memory
LDRH    W13, [X12,#6]         ; Load from Memory
MOV     W14, #0xFFCF          ; Rd = Op2
AND     W13, W13, W14         ; Rd = Op1 & Op2
BFI     W13, W11, #4, #2      ; Bit Field Insert
STRH    W13, [X12,#6]         ; Store to Memory
```

```
Adam-MBP16:tmp adam$ cat d5500 | grep 0x1B0 | grep STR
0xfffffff006c43f14L     STR             X8, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f48L     STR             X8, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f54L     STR             XZR, [X19,#0x1B0]; Store to Memory
0xfffffff006c43f60L     STR             XZR, [X19,#0x1B0]; Store to Memory
0xfffffff006c46108L     STRB            WZR, [X19,#0x1B0]; Store to Memory
Adam-MBP16:tmp adam$
```

CH264Decoder::DecodeStream error h264fw_SetPpsAndSps

- Can it be PPS\SPS?
- Decode a video, iOS kernel tracing, and check
- Take a random H.264 AVC video and analyze it
  - Plenty of tools in GitHub
- Check 0x1B0 to see if it looks like the SPS we sent
- Match!

- This is sufficient to understand the mysterious check!
- SPS->chroma_format_idc == 0 -> kernel overflow!
- Just create a video with chroma_format_idc == 0 and try to decode it

- CMVideoFormatDescriptionCreateFromH264ParameterSets
  - This initializes the session with the requested PPS and SPS

- VTDecompressionSessionCreate
  - Initializes our session with mediaserverd
    - Requires the output from above
- VTDecompressionSessionDecodeFrame
- And…

- Nothing happens!
- Reversing mediaserverd...
- Mediaserverd checks that chroma_format_idc > 0
  - And denies chroma_format_idc == 0

H.264 format

- Reading the H.264 format reveals:

7.3.3 Slice header syntax

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
if(separate_colour_plane_flag = = 1)		
colour_plane_id	2	u(2)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(IdrPicFlag)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(bottom_field_pic_order_in_frame_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type = = 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0]	2	se(v)

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type == 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup]	1	ue(v)
else if(slice_group_map_type == 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup]	1	ue(v)
bottom_right[iGroup]	1	ue(v)
}		
else if(slice_group_map_type == 3 ||		

7.3.2.1.1 Sequence parameter set data syntax

seq_parameter_set_data( ) {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
constraint_set4_flag	0	u(1)
constraint_set5_flag	0	u(1)
reserved_zero_2bits /* equal to 0 */	0	u(2)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 100 || profile_idc == 110 || profile_idc == 122 || profile_idc == 244 || profile_idc == 44 || profile_idc == 83 || profile_idc == 86 || profile_idc == 118 || profile_idc == 128 || profile_idc == 138 || profile_idc == 139 || profile_idc == 134 || profile_idc == 135) {		
chroma_format_idc	0	ue(v)

7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation

This clause specifies the activation process of picture and sequence parameter sets for coded video sequences that conform to one or more of the profiles specified in Annex A and are decoded using the decoding process specified in clauses 2 to 9.

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units or coded slice data partition A NAL units of one or more coded pictures. Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular picture parameter set RBSP results in the deactivation of the previously-active picture parameter set RBSP (if any).

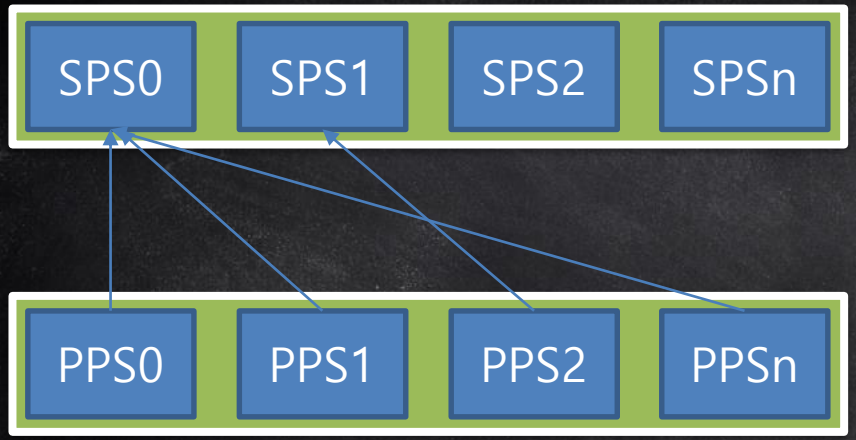
When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not active and it is referred to by a coded slice NAL unit or coded slice data partition A NAL unit (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

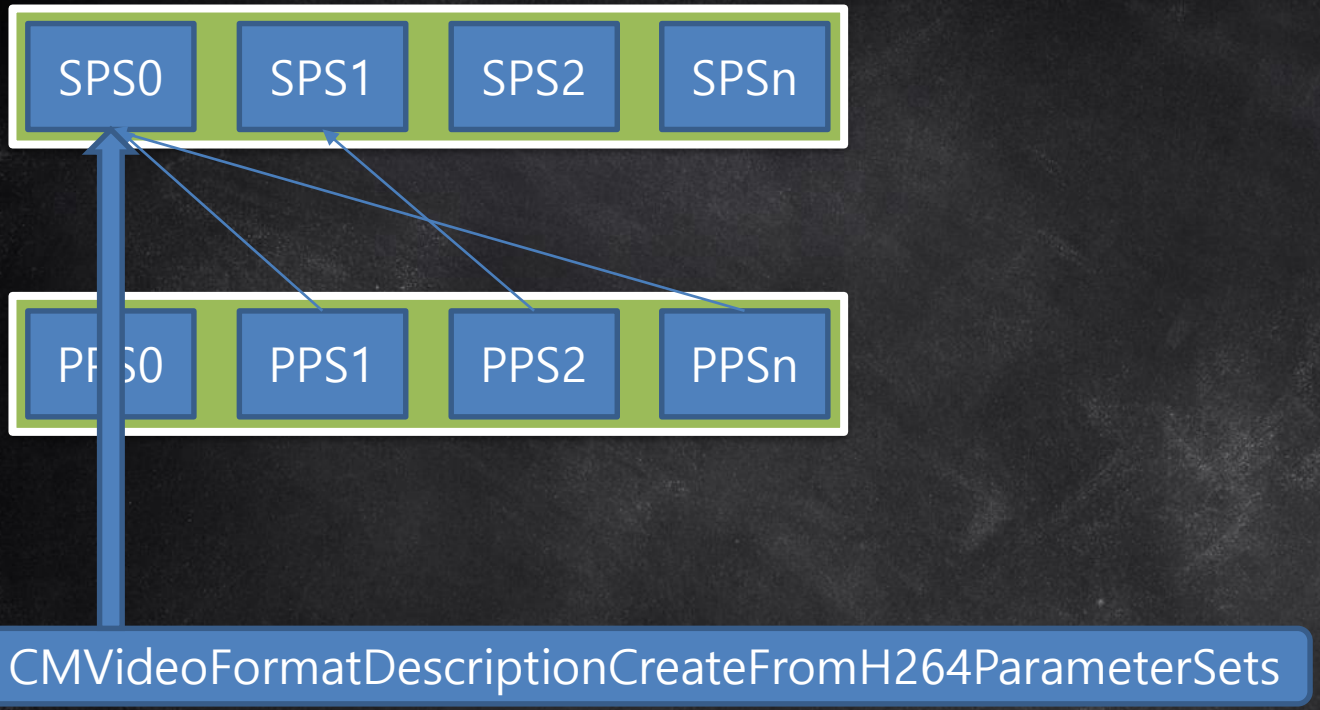
Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active picture parameter set RBSP for a coded picture shall have the same content as that of the active picture parameter set RBSP for the coded picture unless it follows the last VCL NAL unit of the coded picture and precedes the first VCL NAL unit of another coded picture.

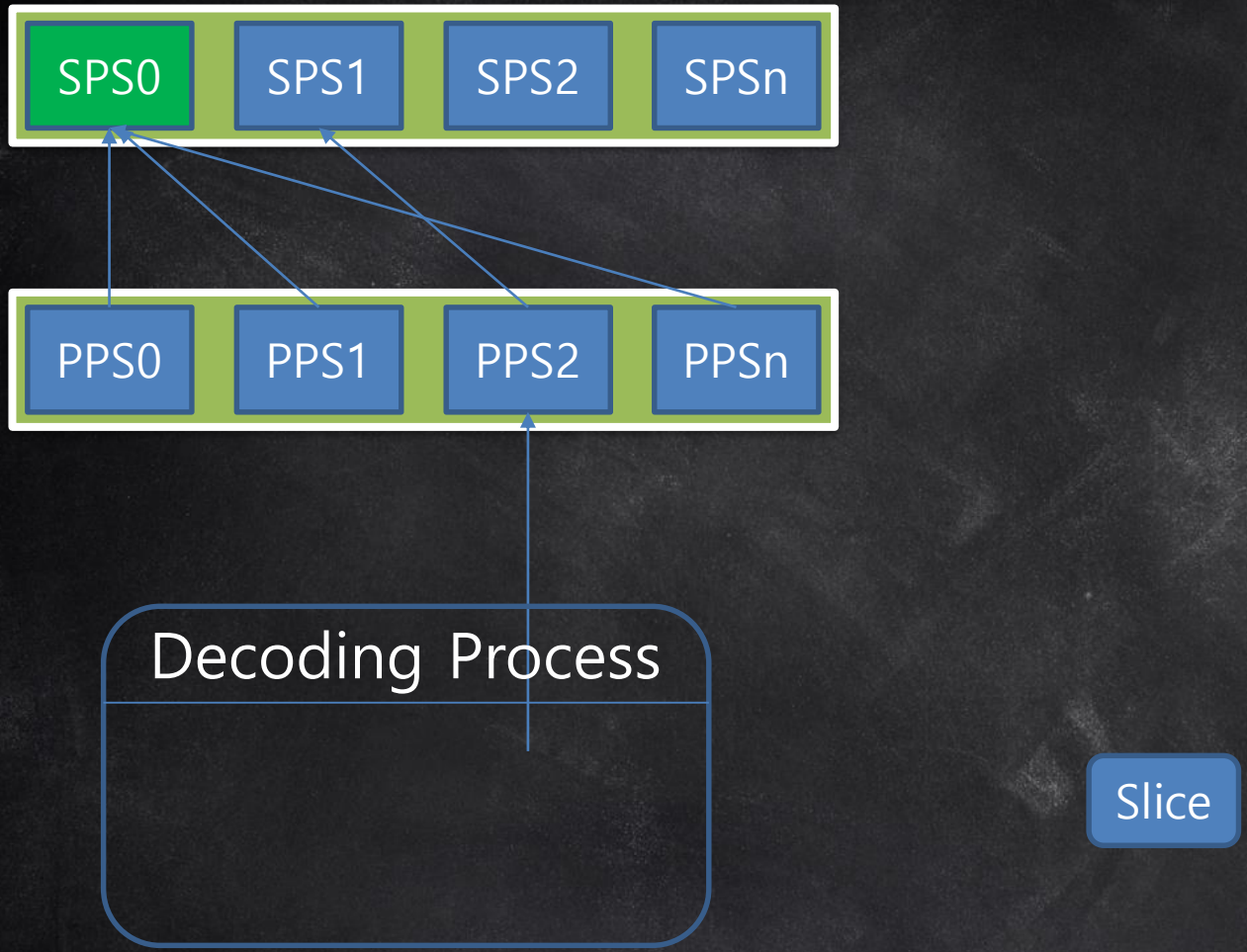
When a picture parameter set NAL unit with a particular value of pic_parameter_set_id is received, its content replaces the content of the previous picture parameter set NAL unit, in decoding order, with the same value of pic_parameter_set_id (when a previous picture parameter set NAL unit with the same value of pic_parameter_set_id was present in the bitstream).

NOTE 2 – A decoder must be capable of simultaneously storing the contents of the picture parameter sets for all values of pic_parameter_set_id. The content of the picture parameter set with a particular value of pic_parameter_set_id is overwritten when a new picture parameter set NAL unit with the same value of pic_parameter_set_id is received.

A sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more SEI NAL units containing a buffering period SEI message. Each sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one sequence parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular sequence parameter set RBSP results in the deactivation of the previously-active sequence parameter set RBSP (if any).


SPS0
SPS1
SPS2
SPSn
PPS0
PPS1
PPS2
PPSn


SPS0
SPS1
SPS2
SPSn
PPS0
PPS1
PPS2
PPSn
CMVideoFormatDescriptionCreateFromH264ParameterSets


SPS0
SPS1
SPS2
SPSn
PPS0
PPS1
PPS2
PPSn
Decoding Process
Slice

H.264 format

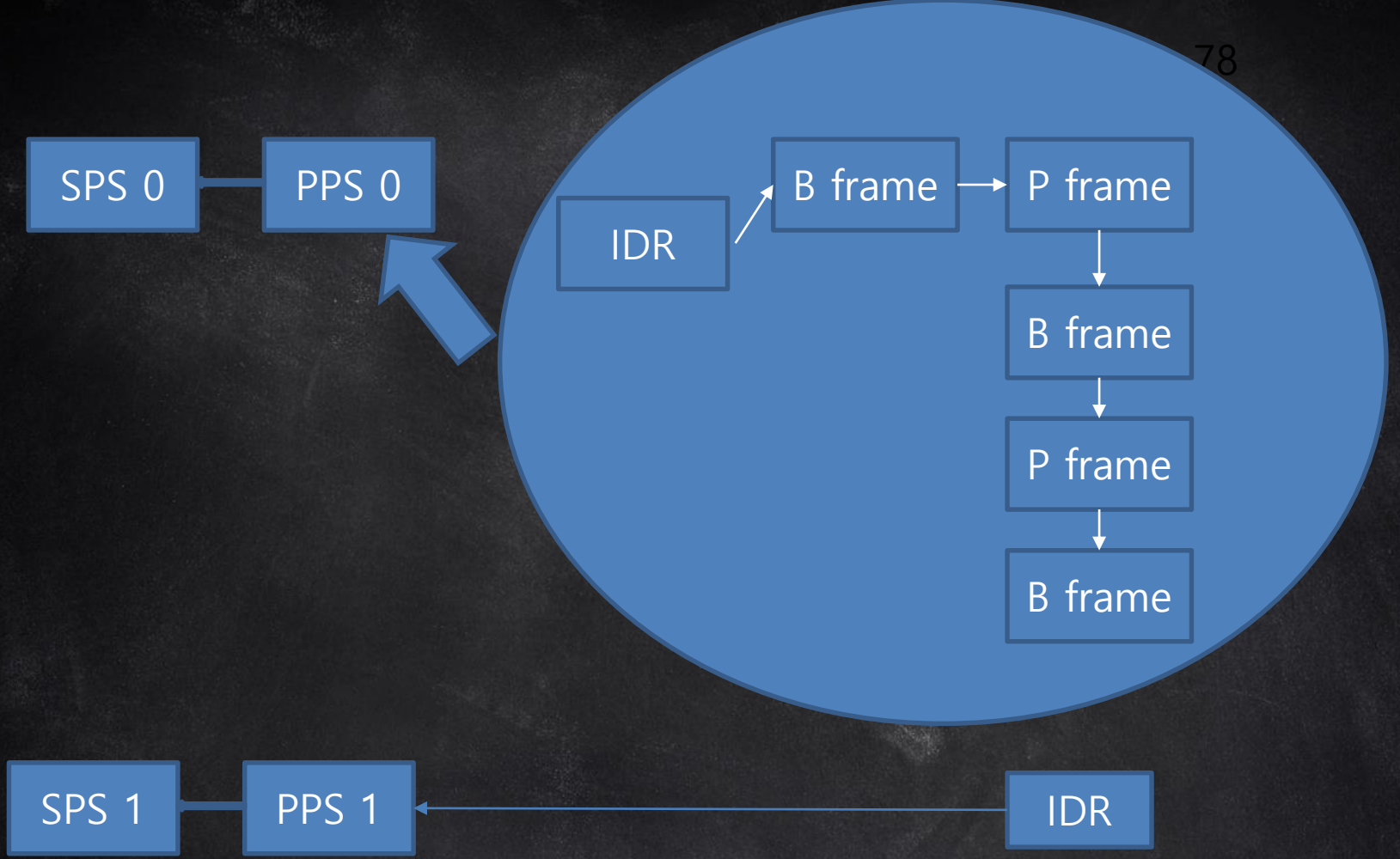
Sequence parameter set

- Can there be multiple sequence parameter set NALs?
- CMVideoFormatDescriptionCreateFromH264ParameterSets is only called once for mediaserverd
- Nevertheless…

H.264 format

- C
- Cr
- Cre
- Sen

AL


SPS 0
PPS 0
IDR
B frame
P frame
B frame
P frame
B frame
SPS 1
PPS 1
IDR


11

AppleD5500

- Same code doesn’t crash it anymore…
- No apparent change in AppleD5500…
- H264H8.videodecoder is changed
- “canvasSurfaceID” no longer appears in the strings
- Apple separated between decoding and tile decoding

Debugging iOS processes

- Assuming kernel RW

- *Debugserver 0.0.0.0:1234 –a mediaserverd*
- Doesn’t work

2. Save the following as ent.xml:

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.springboard.debugapplications</key>
    <true/>
    <key>get-task-allow</key>
    <true/>
    <key>task_for_pid-allow</key>
    <true/>
    <key>run-unsigned-code</key>
    <true/>
</dict>
</plist>
```

debugserver

- Gi
- La
- Att

```
(lldb) process connect connect://localhost:1234
Process 29 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
    frame #0: 0x00000001857c4bc4 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
->  0x1857c4bc4 <+8>: ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
    0x1857c4bc8 <+0>: mov    x16, #-0x20
    0x1857c4bcc <+4>: svc    #0x80
    0x1857c4bd0 <+8>: ret
Target 0: (mediaserverd) stopped.
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
  * frame #0: 0x00000001857c4bc4 libsystem_kernel.dylib`mach_msg_trap + 8
    frame #1: 0x00000001857c4a3c libsystem_kernel.dylib`mach_msg + 72
    frame #2: 0x0000000185c75c74 CoreFoundation`__CFRunLoopServiceMachPort + 196
    frame #3: 0x0000000185c73840 CoreFoundation`__CFRunLoopRun + 1424
    frame #4: 0x0000000185b93fb8 CoreFoundation`CFRunLoopRunSpecific + 436
    frame #5: 0x00000001005e4518 mediaserverd`_mh_execute_header + 17688
    frame #6: 0x00000001856b6568 libdyld.dylib
(lldb) register read
General Purpose Registers:
        x0 = 0x0000000010004005
        x1 = 0x0000000007000806
        x2 = 0x0000000000000000
```

iOS 11 modifications

- A
- Ne
- Deb

iOS 11 modifications


```
OSStatus
VTTileDecompressionSessionDecodeTile(
        CM_NONNULL VTDecompressionSessionRef     session,
        CM_NONNULL CMSampleBufferRef             sampleBuffer,
        VTDecodeFrameFlags                       decodeFlags,
        void * CM_NULLABLE                       sourceFrameRefCon,
        CVPixelBufferRef                         iosurface_buffer,
        uint64_t                                 x_and_y,
        void * CM_NULLABLE                       some_flag,
        VTDecodeInfoFlags * CM_NULLABLE          infoFlagsOut);
```

Disclosure

CVE-2018-4109

30th October, 2017

Vulnerability disclosure to Apple

2nd December, 2017

Apple confirmed the vulnerability

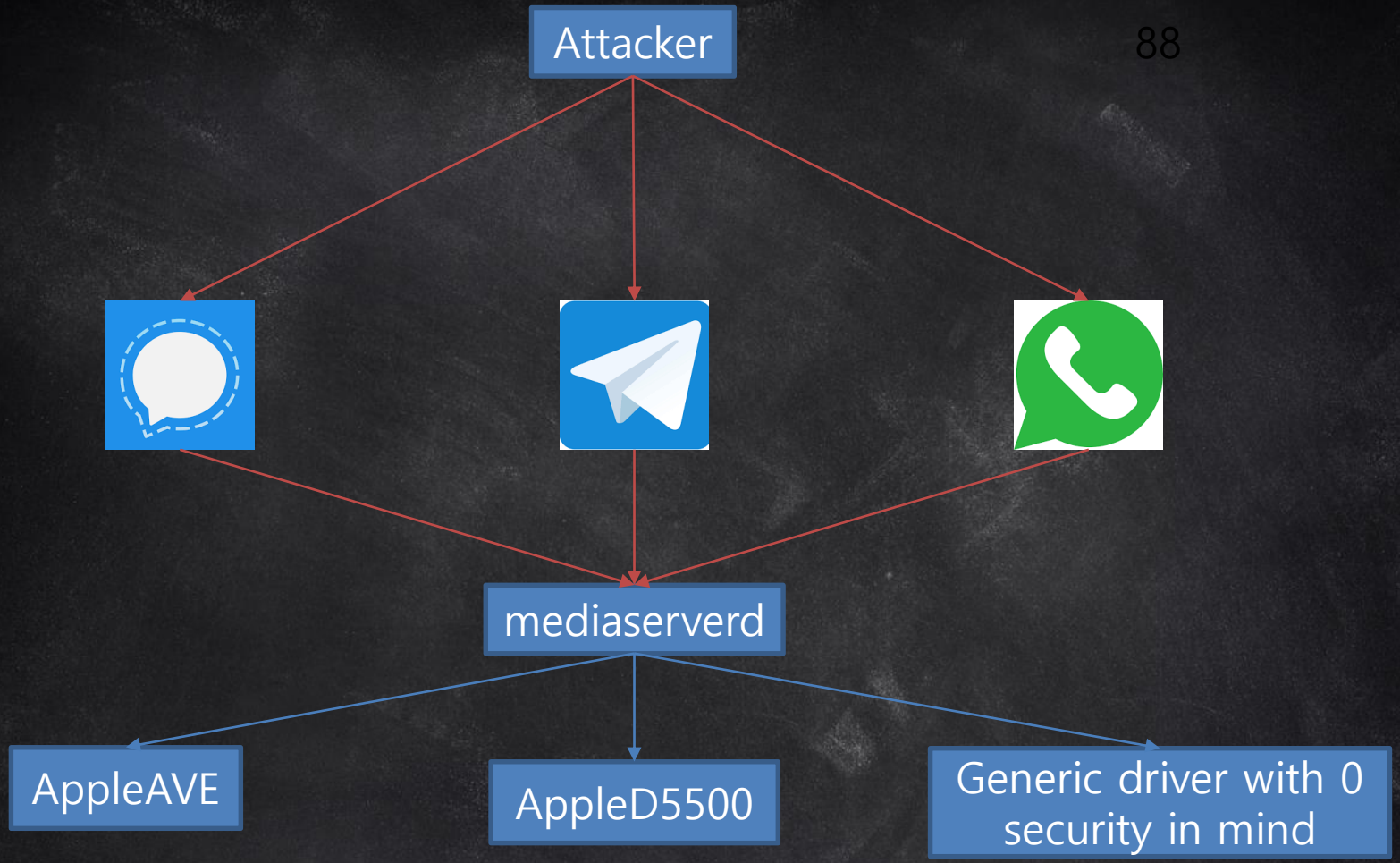
23st January, 2018

Apple deployed the patch to their iDevices

Zero-click vulnerabilities

Mediaserverd is an interesting attack vector

- A lot of code can be triggered in mediaserverd remotely
- For example, video parsing
  - With dozens of different formats and types
- Can be triggered via interesting common apps
  - iMessage, WeChat, WhatsApp, Telegram, Signal, etc…
- Multiple vulnerabilities that ultimately lead to RCE
  - Sometimes within mediaserverd
    - Mediaserverd’s sandbox is awesome ;)


Attacker
mediaserverd
AppleAVE
AppleD5500
Generic driver with 0 security in mind

Zero-click vulnerabilities

Mediaserverd ~~is~~ was an interesting attack vector

- With the introduction of PAC, zero clicks are dead
  - Along with ROPs and most of the JOPs

```
LDR     X8, [X21, #0]
LDR     X8, [X8, #0x28]
MOV     X0, X21
BLR     X8
```



```
LDR     X8, [X21, #0]
LDRAA   X9, [X8, #8]!
MOVK    X8, 0x3a87, LSL 48
MOV     X0, X21
BLRAA   X8
```

Takeaways

- Manuals are useful
  - Even if you hate them
- Infrastructure work saves a lot of time
- Hit them where it hurts the most
  - Nobody looks at AppleD5500
  - Find the most minimal access to such drivers and you're back at 2007

- iOS still has a way to go

Thank you!