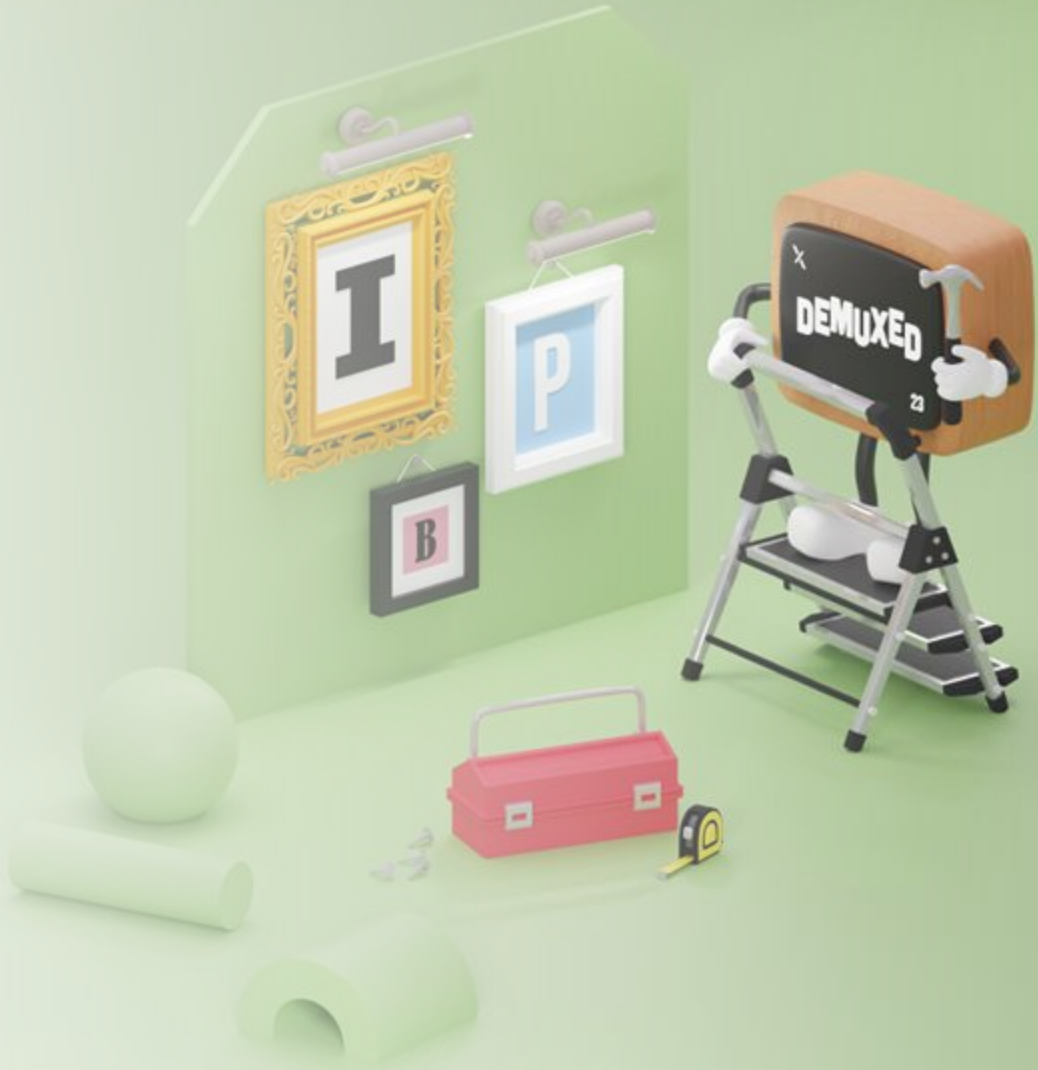
Having H26Fun with H26Forge:
Vulnerability Hunting, Datamoshing, and More!

Willy R. Vasquez
wrv@utexas.edu

October 25th, 2023
Demuxed 2023



$whoami

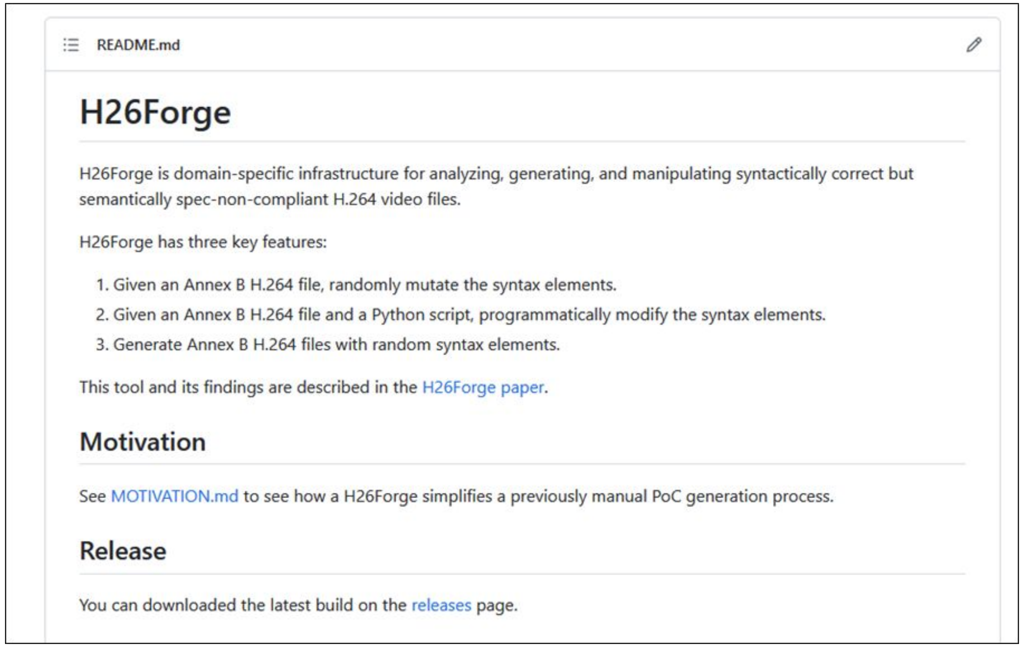
- Willy R. Vasquez (wrv)
- PhD Student at UT Austin
- Systems Security, Cryptography, and Cyber Law and Policy
- Long time Demuxed viewer, first time attendee and presenter

https://wrv.github.io/

Used H26Forge to find and report issues in applications, kernel drivers, and hardware

CVE-2022-48434, CVE-2022-42850, CVE-2022-42846, CVE-2022-32939, CVE-2022-3266

H26Forge is a Toolkit to work with H.264 Syntax Elements


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.

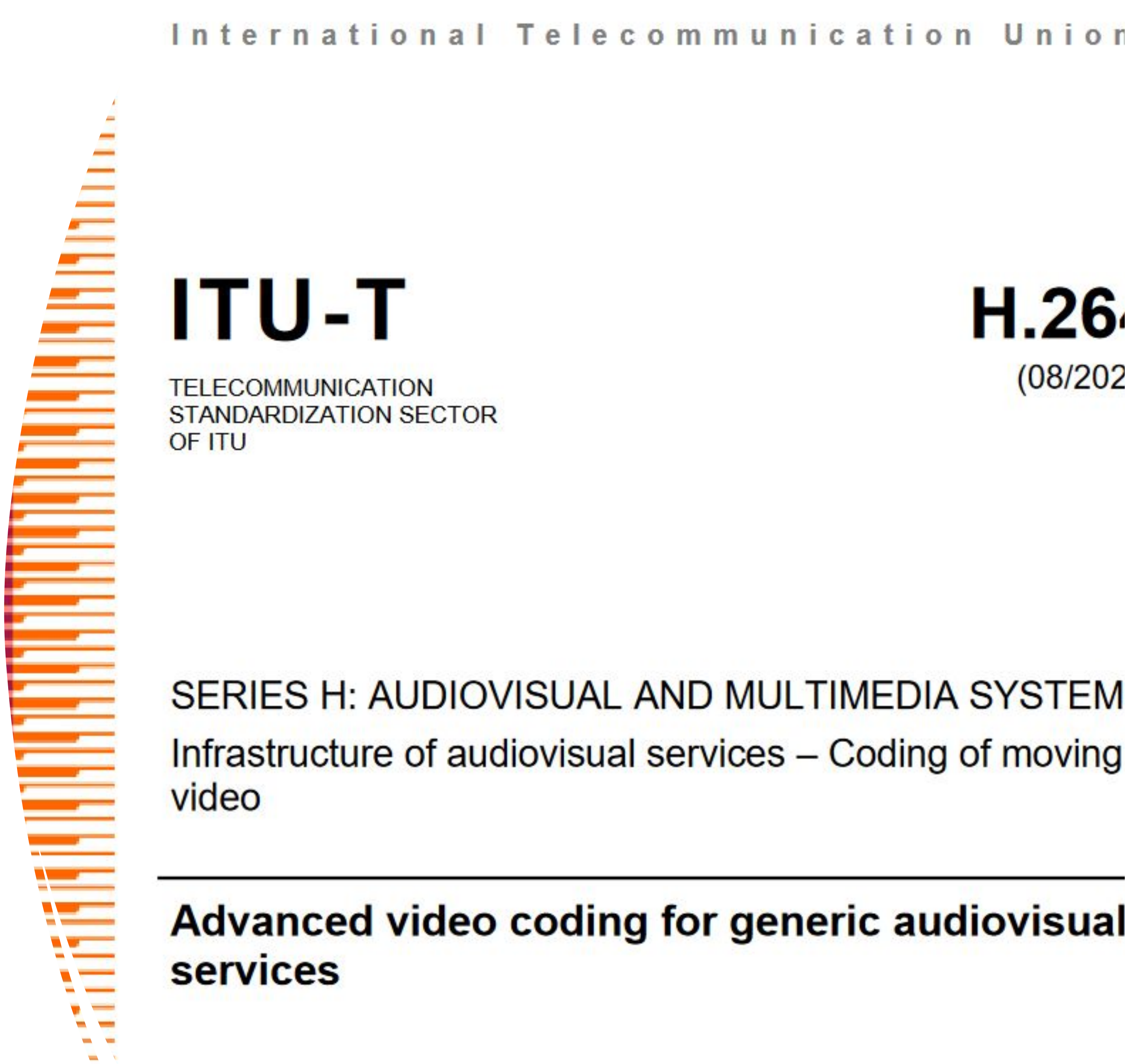
Release

You can downloaded the latest build on the releases page.

https://github.com/h26forge/h26forge

H.264/AVC

- Standardized in 2004
- Over 800-page spec
- Spec only specifies decoding
- Ubiquitous device support



International Telecommunication Unio

ITU-T

H.26

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(08/202

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEM

Infrastructure of audiovisual services – Coding of moving
video

**Advanced video coding for generic audiovisual
services**

H.264 Syntax Elements

log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
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)
}		
max_num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Syntax Elements

Decoding Instructions read from the Bitstream

log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
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)
}		
max_num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Syntax Elements

Decoding Instructions read from the Bitstream

log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
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)
}		
max_num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Semantics

How the decoding instruction is used and the expected range

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\,\text{log2\_max\_frame\_num\_minus4} + 4\,)} \qquad (7\text{-}10)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in clause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following:

- an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,
- two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture,
- an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in clause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\,\text{log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4\,)} \qquad (7\text{-}11)$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Semantics

How the decoding instruction is used and the expected range

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{log2\_max\_frame\_num\_minus4} + 4)} \quad (7\text{-}10)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in clause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following:

– an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,

– two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture,

– an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in clause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4)} \quad (7\text{-}11)$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Semantics

How the decoding instruction is used and the expected range

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{log2\_max\_frame\_num\_minus4} + 4)} \qquad (7\text{-}10)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in clause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following:

– an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,

– two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture,

– an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in clause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{log2\_max\_pic\_order\_cnt\_lsb\_minus4} + 4)} \qquad (7\text{-}11)$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Entropy Encoding

The bitstream representation

log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
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)
}		
max_num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

https://www.itu.int/rec/T-REC-H.264-202108-I/en

H.264 Entropy Encoding

The bitstream representation

log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
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)
}		
max_num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

https://www.itu.int/rec/T-REC-H.264-202108-I/en

Goal: Modify H.264 Syntax Elements

Goal: Modify H.264 Syntax Elements

- Use cases

Goal: Modify H.264 Syntax Elements

- Use cases
  - Vulnerability Hunting

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

Natalie Silvanovich

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

Goal: Modify H.264 Syntax Elements

- Use cases
  - Vulnerability Hunting
  - Datamoshing

https://i.kym-cdn.com/photos/images/original/000/475/486/0e9.gif

Goal: Modify H.264 Syntax Elements

- Use cases
  - Vulnerability Hunting
  - Datamoshing
- Existing Tools?

Goal: Modify H.264 Syntax Elements

- Use cases
  - Vulnerability Hunting
  - Datamoshing
- Existing Tools?
  - FFmpeg CodedBitstream (CBS): select syntax elements (SEI, AUD, VUI)

Goal: Modify H.264 Syntax Elements

- Use cases
  - Vulnerability Hunting
  - Datamoshing
- Existing Tools?
  - FFmpeg CodedBitstream (CBS): select syntax elements (SEI, AUD, VUI)
  - Bitstream viewers: primarily used to tune an encoder

Goal: Modify H.264

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

Natalie Silvanovich

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

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

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

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

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)

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

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)

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

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)

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)

Set to 0

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	01	67	64	00	0B	AC	A7	28	49	BF	10	80	

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		

Set to 0

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	01	67	64	00	0B	AC	A7	28	49	BF	10	80	

Set to 1

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	01	67	64	00	0B	AC	A6	A5	09	37	E2	10	

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		

	Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Set to 0	00000000	00	00	00	01	67	64	00	0B	AC	A7	28	49	BF	10	80	

	Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Set to 1	00000000	00	00	00	01	67	64	00	0B	AC	A6	A5	09	37	E2	10	

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)

Set to 0

0xA728: 0b1010 0111 0010 1000

09	0A	0B	0C	0D	0E	0F
A7	28	49	BF	10	80	

Set to 1

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	01	67	64	00	0B	AC	A6	A5	09	37	E2	10	

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)

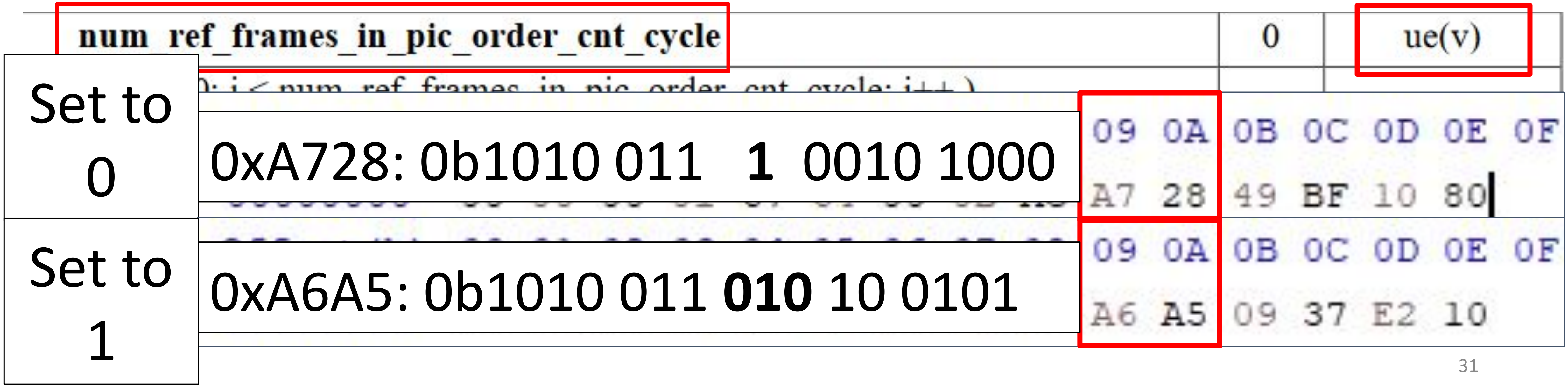
Set to 0	0xA728: 0b1010 0111 0010 1000	09	0A	0B	0C	0D	0E	0F
		A7	28	49	BF	10	80	
Set to 1	0xA6A5: 0b1010 0110 1010 0101	09	0A	0B	0C	0D	0E	0F
		A6	A5	09	37	E2	10	

Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

1. Variable bit-length bitstream representation

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010



Modifying Syntax Elements at the Entropy-Encoded Level

Two key challenges:

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

Decimal Value	Binary Exp-Golomb Encoding
0	1
1	010

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)

Set to 1

0xA6A5: 0b1010 011 **010 1** 0 0101

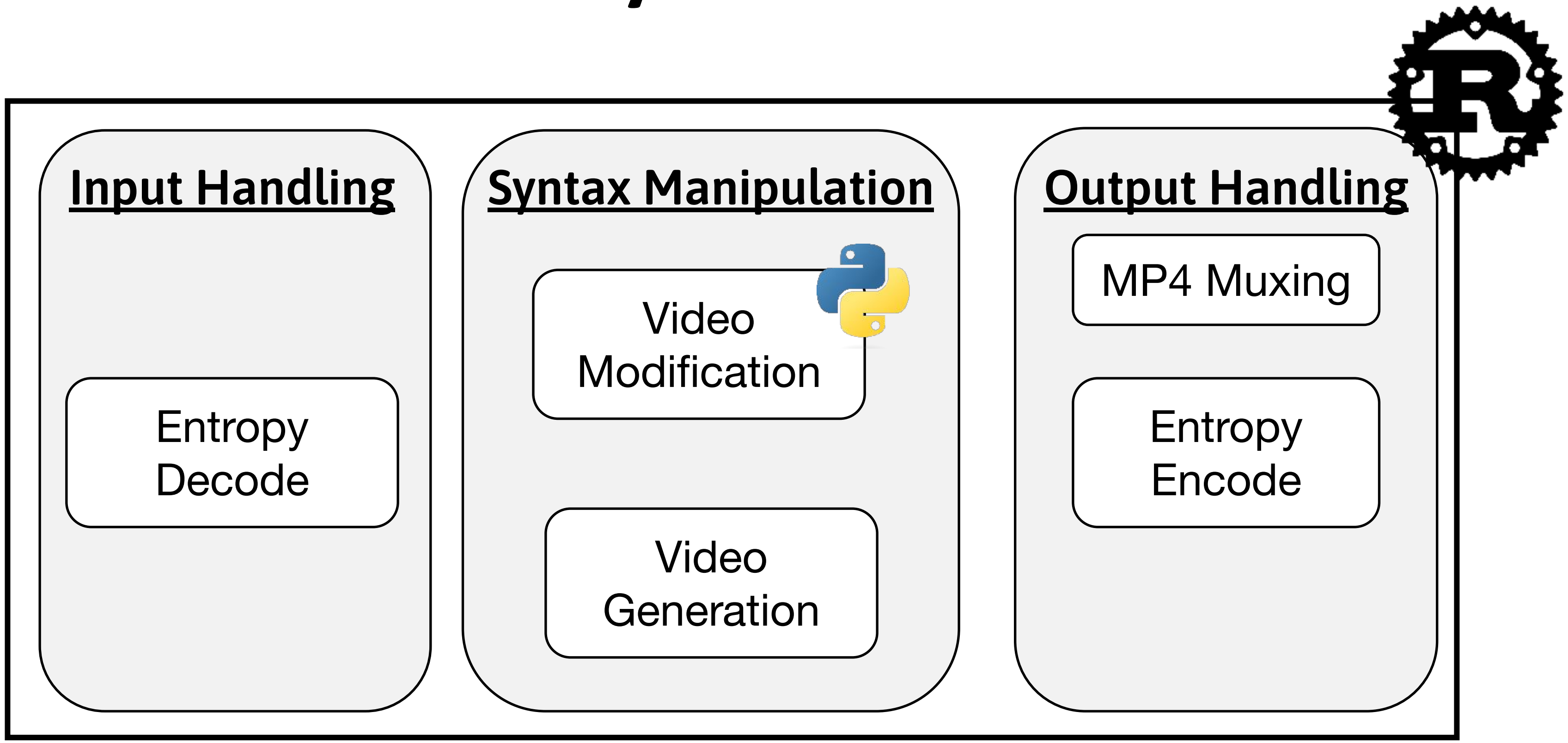
09	0A	0B	0C	0D	0E	0F
A6	A5	09	37	E2	10	

With H26Forge, this is just three lines of Python

```python
amount = 1
ds["spses"][0]["num_ref_frames_in_pic_order_cnt_cycle"] = amount
ds["spses"][0]["offset_for_ref_frame"] = [0] * amount
```

H26Forge: Toolkit to manipulate H.264 Syntax Elements

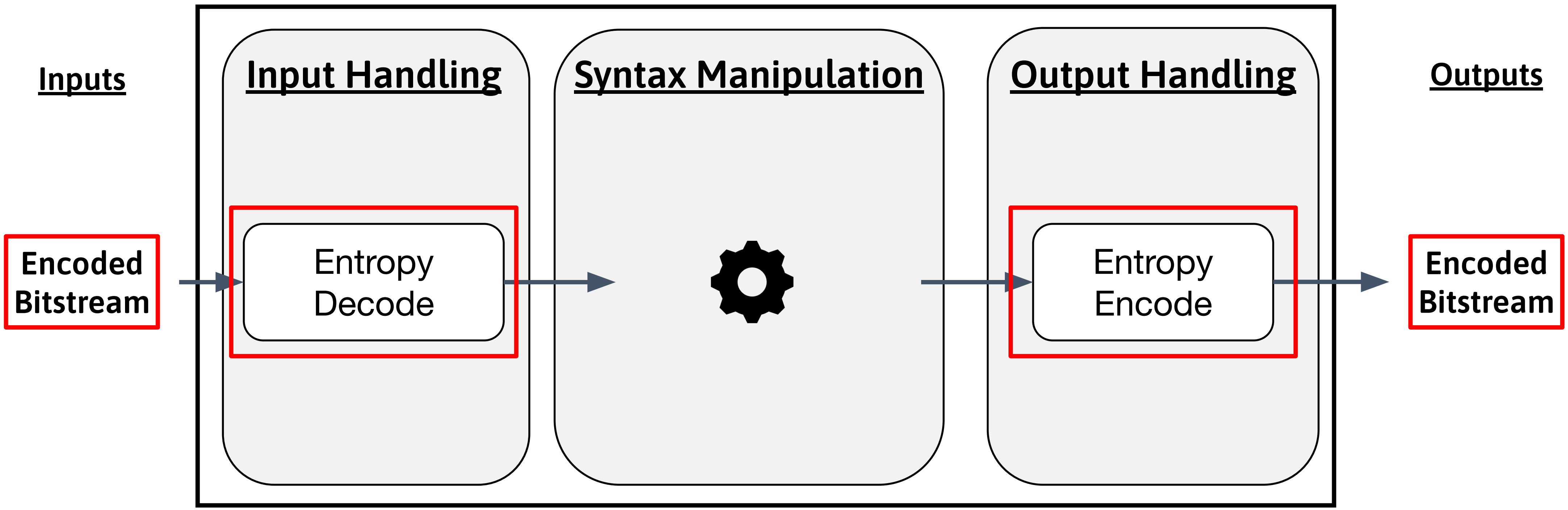
30,000+ lines of Rust



Open Source
MIT License

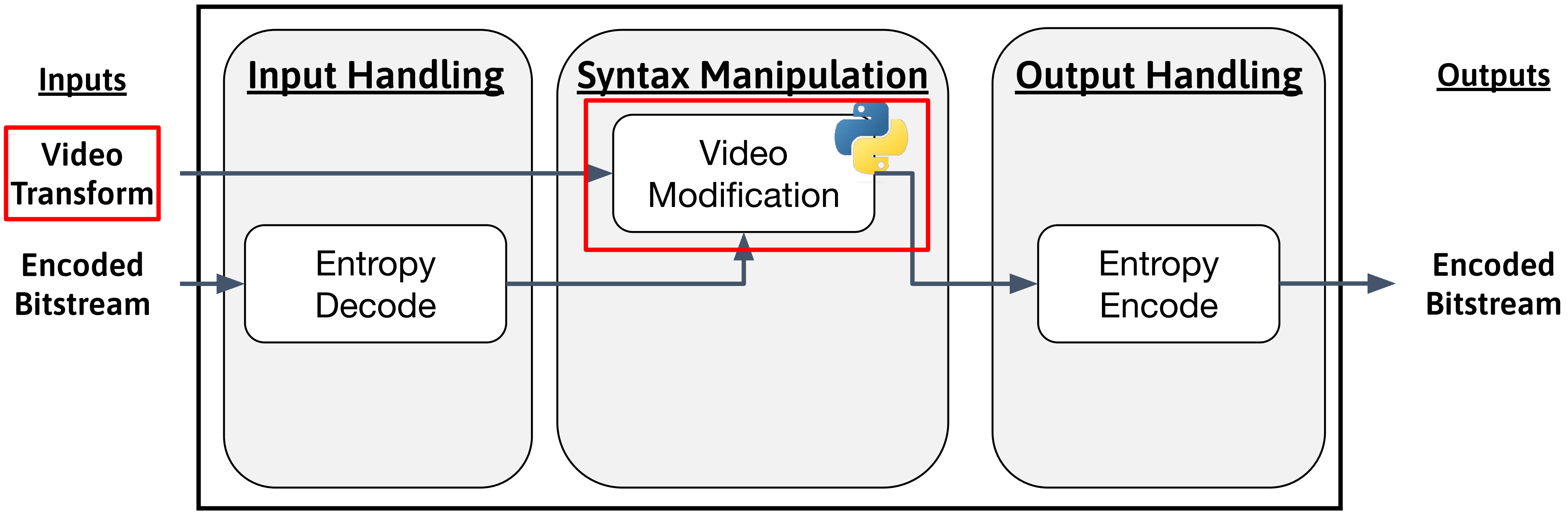
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



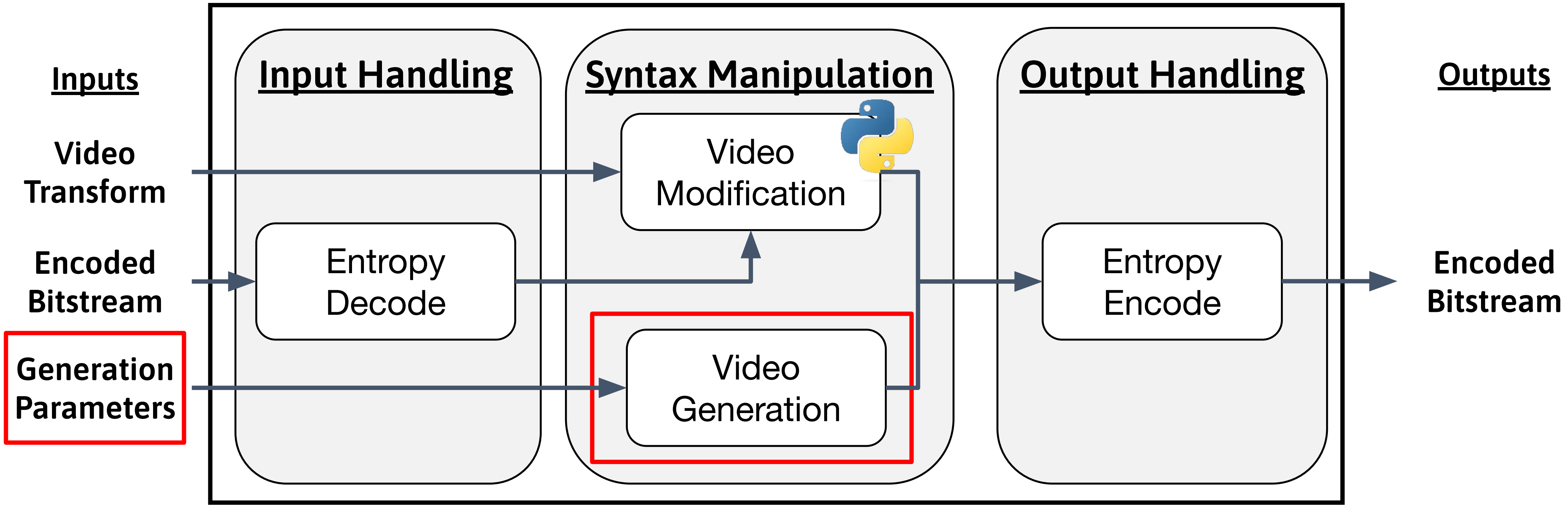
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



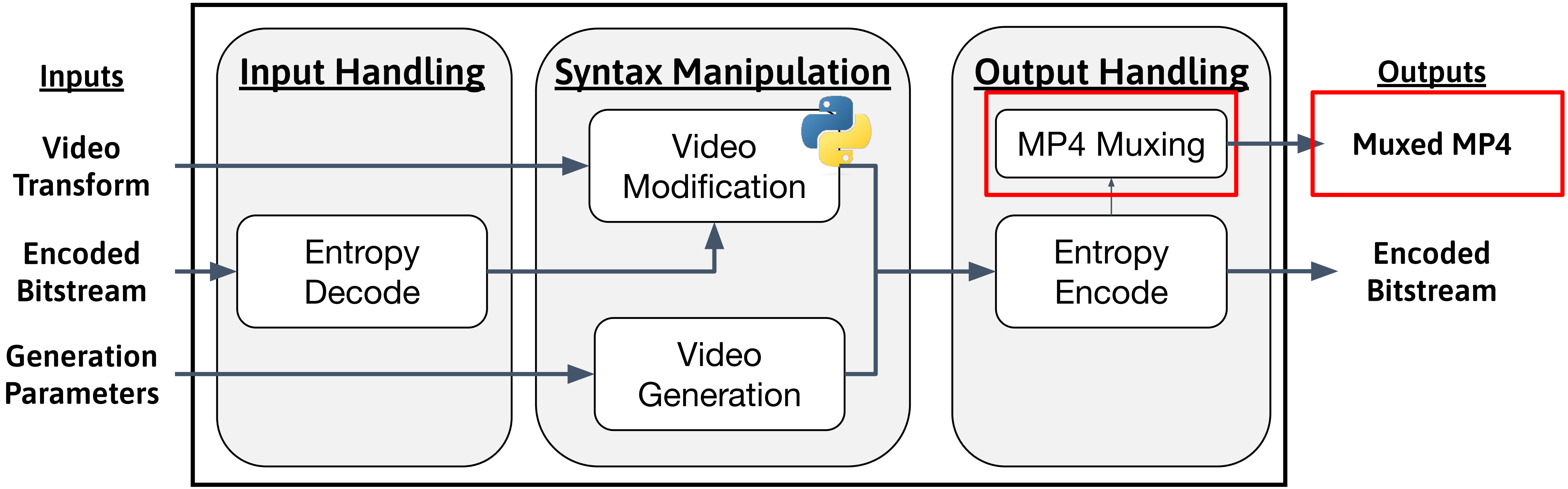
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



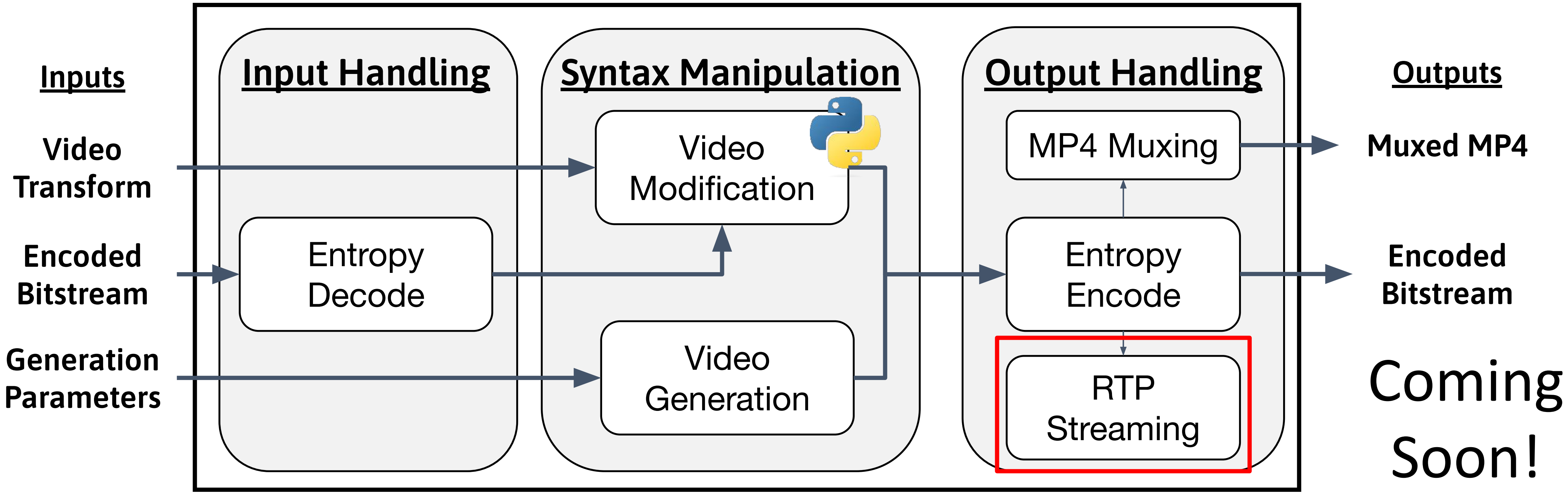
https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



https://github.com/h26forge/h26forge

H26Forge: Toolkit to manipulate H.264 Syntax Elements



https://github.com/h26forge/h26forge

H26Forge Use Cases

H26Forge Use Cases

Vulnerability Hunting

H26Forge Use Cases

Vulnerability Hunting

Datamoshing

https://gifer.com/en/O2FI

H26Forge Use Cases

Vulnerability Hunting

Datamoshing

https://gifer.com/en/O2FI

https://i.imgur.com/s0ga02E.jpeg

Vulnerability Hunting

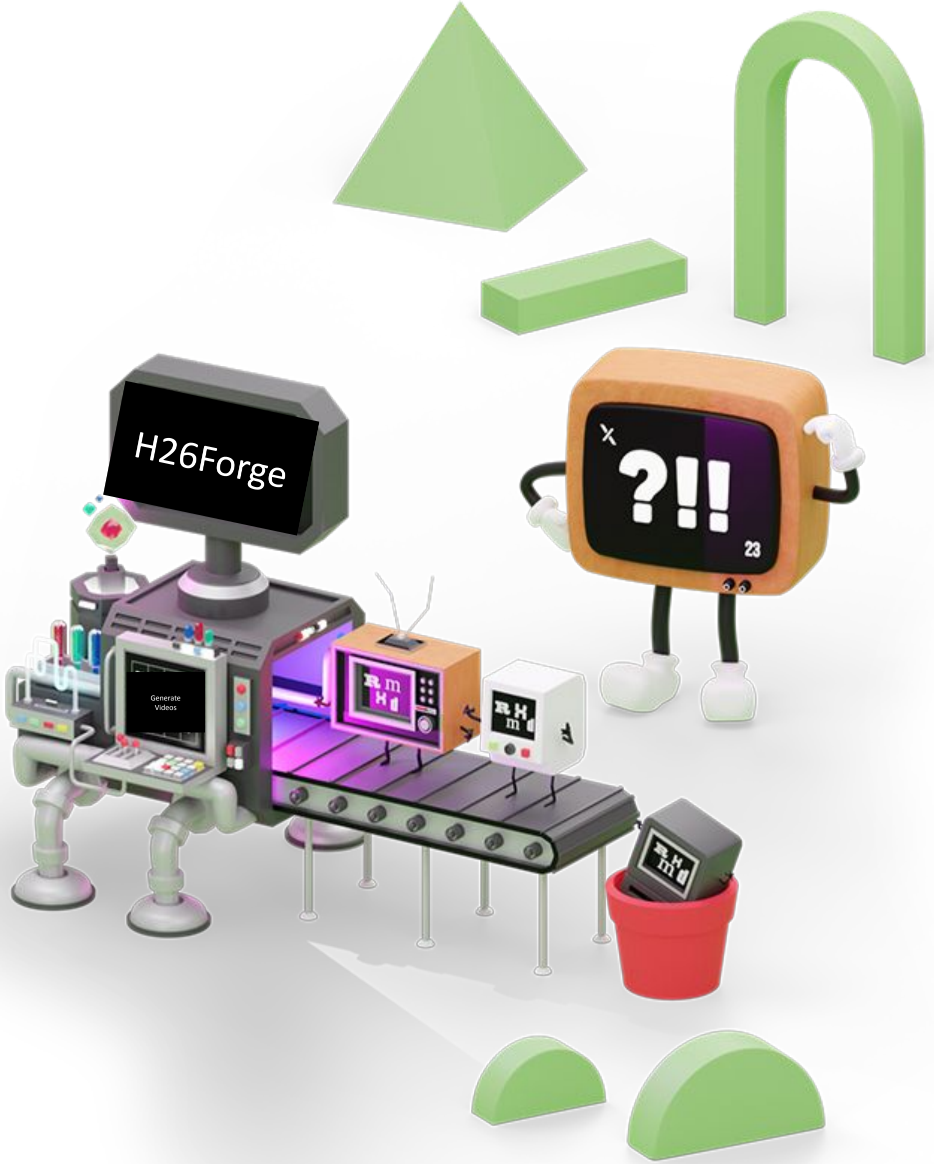
- Decoders may miss semantic checks, leading to undefined states
- Undefined states may have security consequences

Vulnerability Hunting

- Decoders may miss semantic checks, leading to undefined states
- Undefined states may have security consequences

H26Forge can **generate videos with random and out-of-bounds syntax elements** to find security vulnerabilities

H26Forge can **generate a proof-of-concept once a vulnerability** is found

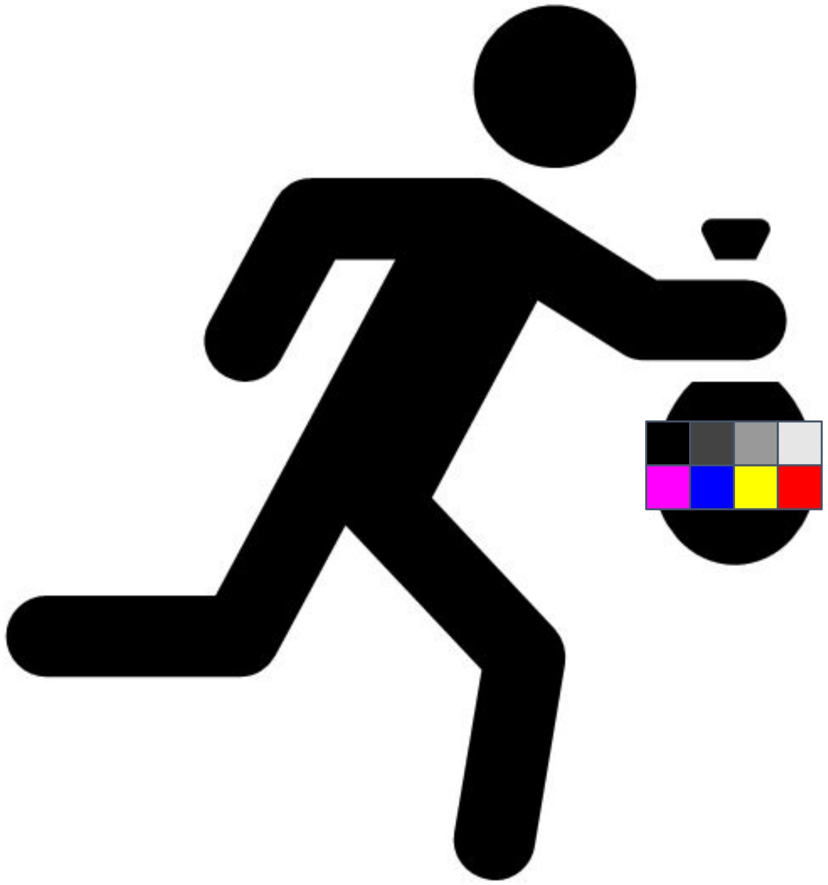


Vulnerabilities We Discovered

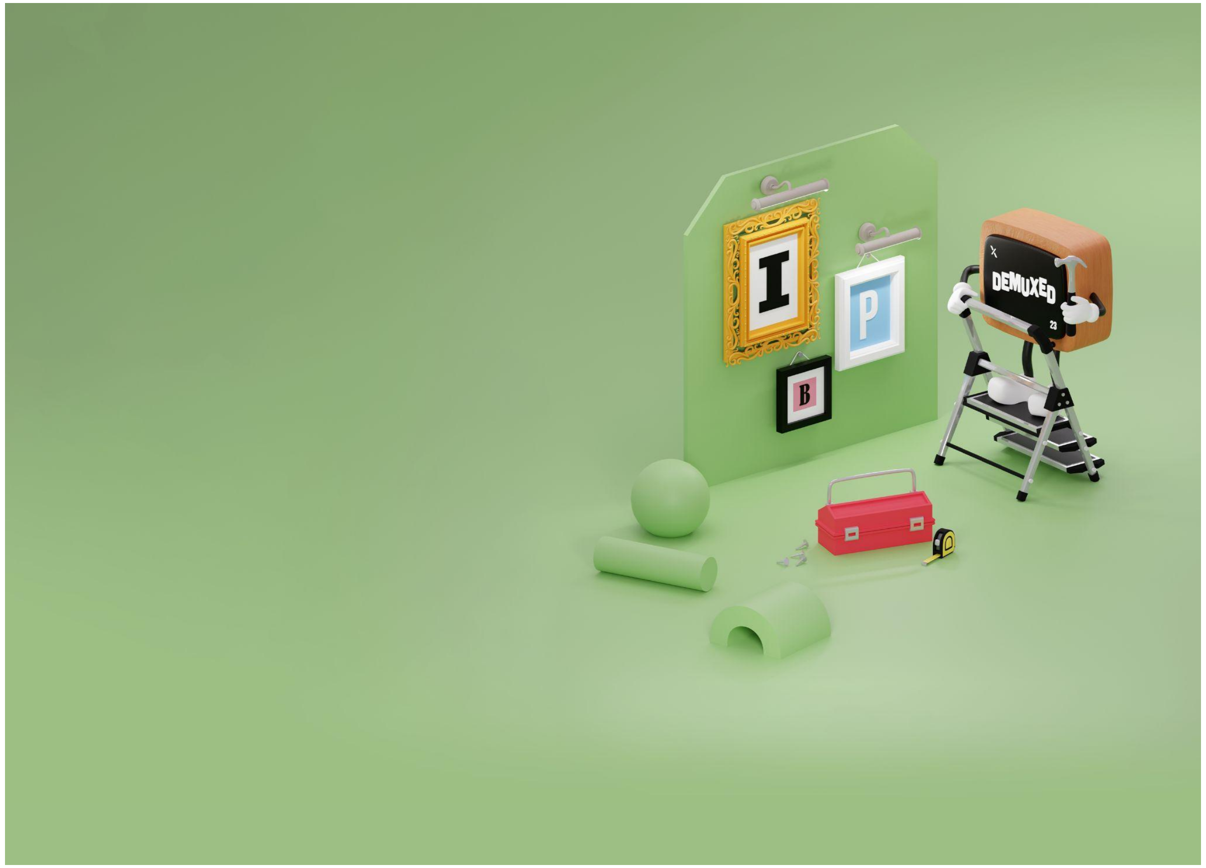
1. **CVE-2022-48434** – Use-after-free in **FFmpeg** as used by **VLC** due to an SPS change mid-video.
2. **CVE-2022-42850** – A lack of bounds-checking in **H.265** SPS StRefPic list parsing leads to an **iOS kernel** heap overflow.
3. **CVE-2022-42846** – An IDR Inter predicted first slice leads to an **iOS kernel** infinite loop during reference picture list modification. **0-clickable.**
4. **CVE-2022-32939** – More than 256 emulation prevention bytes in a correctly encoded H.264 bitstream led to an arbitrary **iOS kernel** write primitive. **0-clickable.**
5. **CVE-2022-3266** – Video width and height was not updated between container and SPS, and across SPSes in **Firefox**. This led to a crash of the Firefox GPU process and an information leak.

Hardware Issue: `Luma/Chroma Thief`

- **Out-of-bounds Intra prediction** leads to stale or uninitialized data disclosure
- CABAC encoded values decoded in hardware
- Different results across hardware decoders

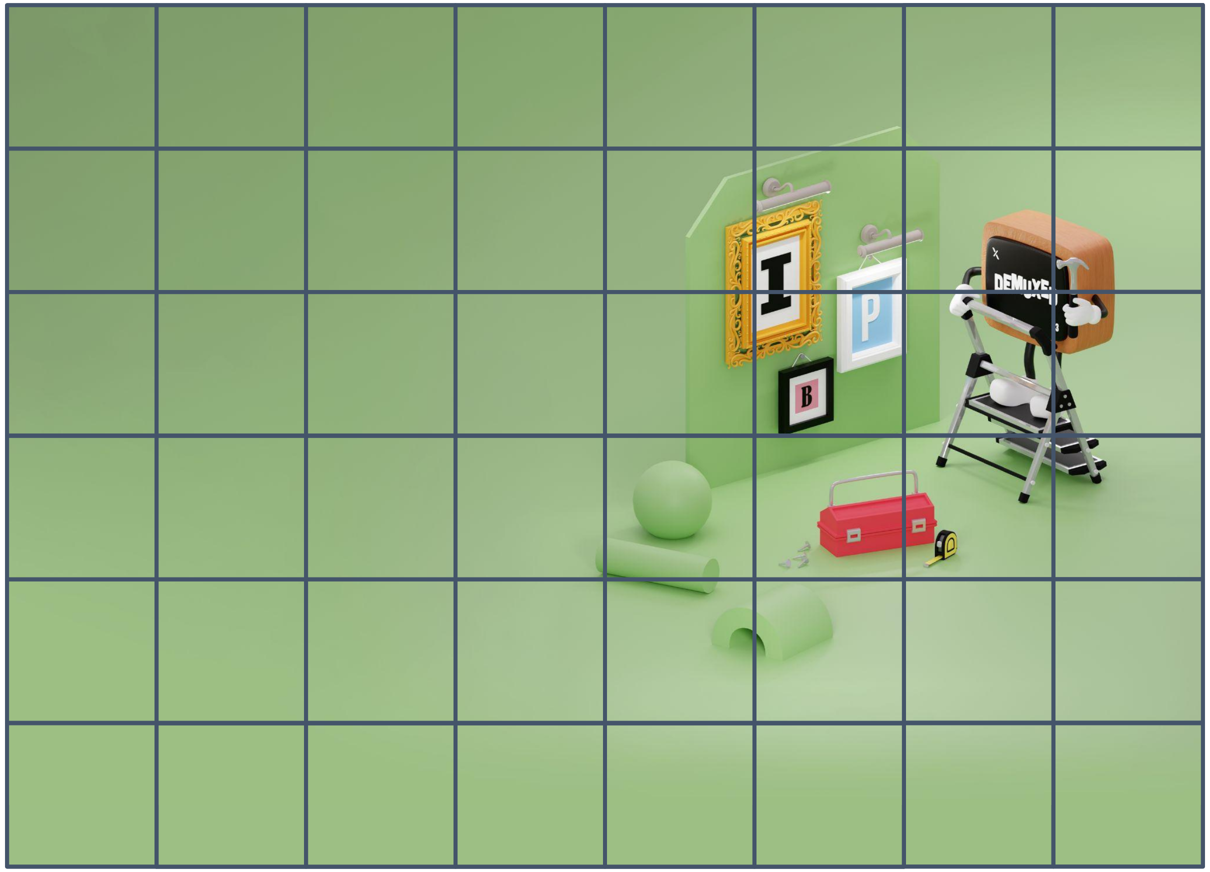


Intra Prediction



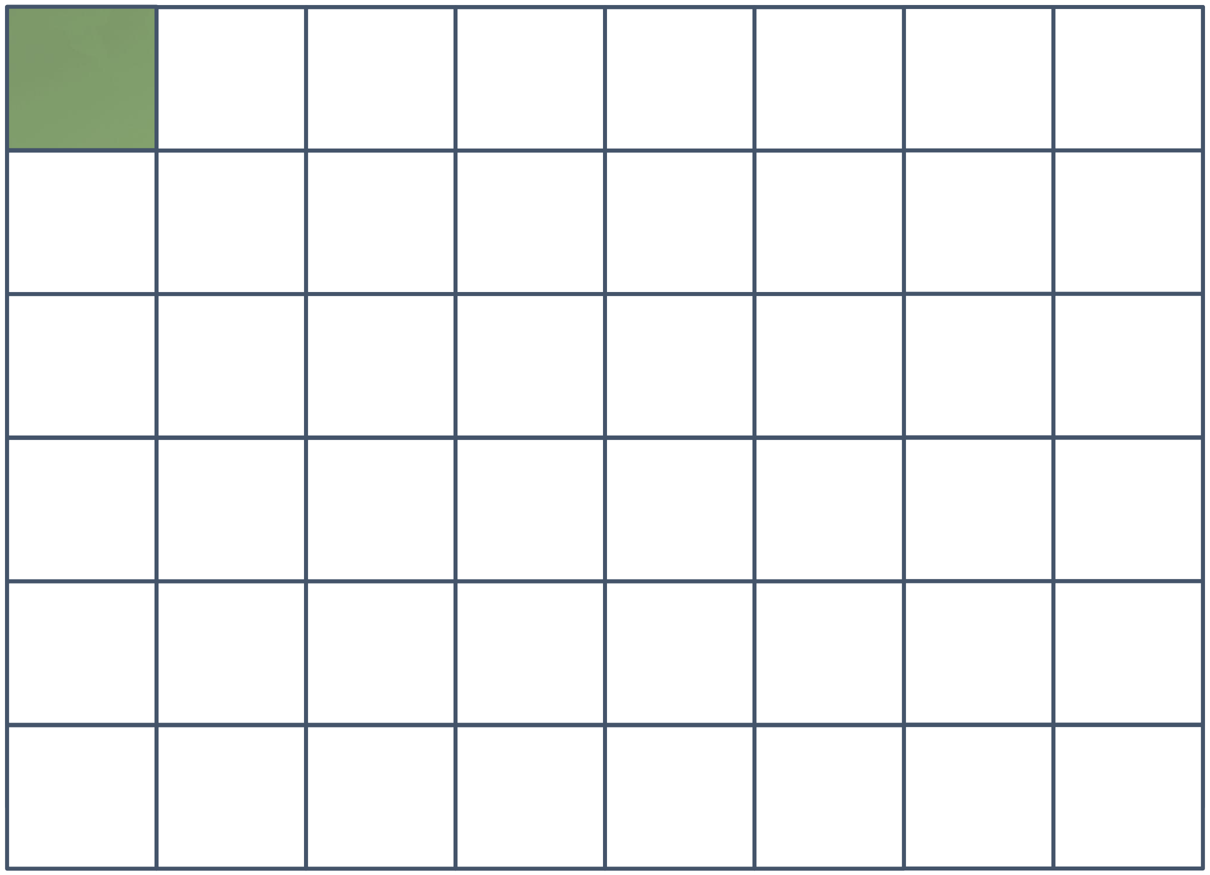
Intra Prediction

Frame is
broken into
Macroblocks



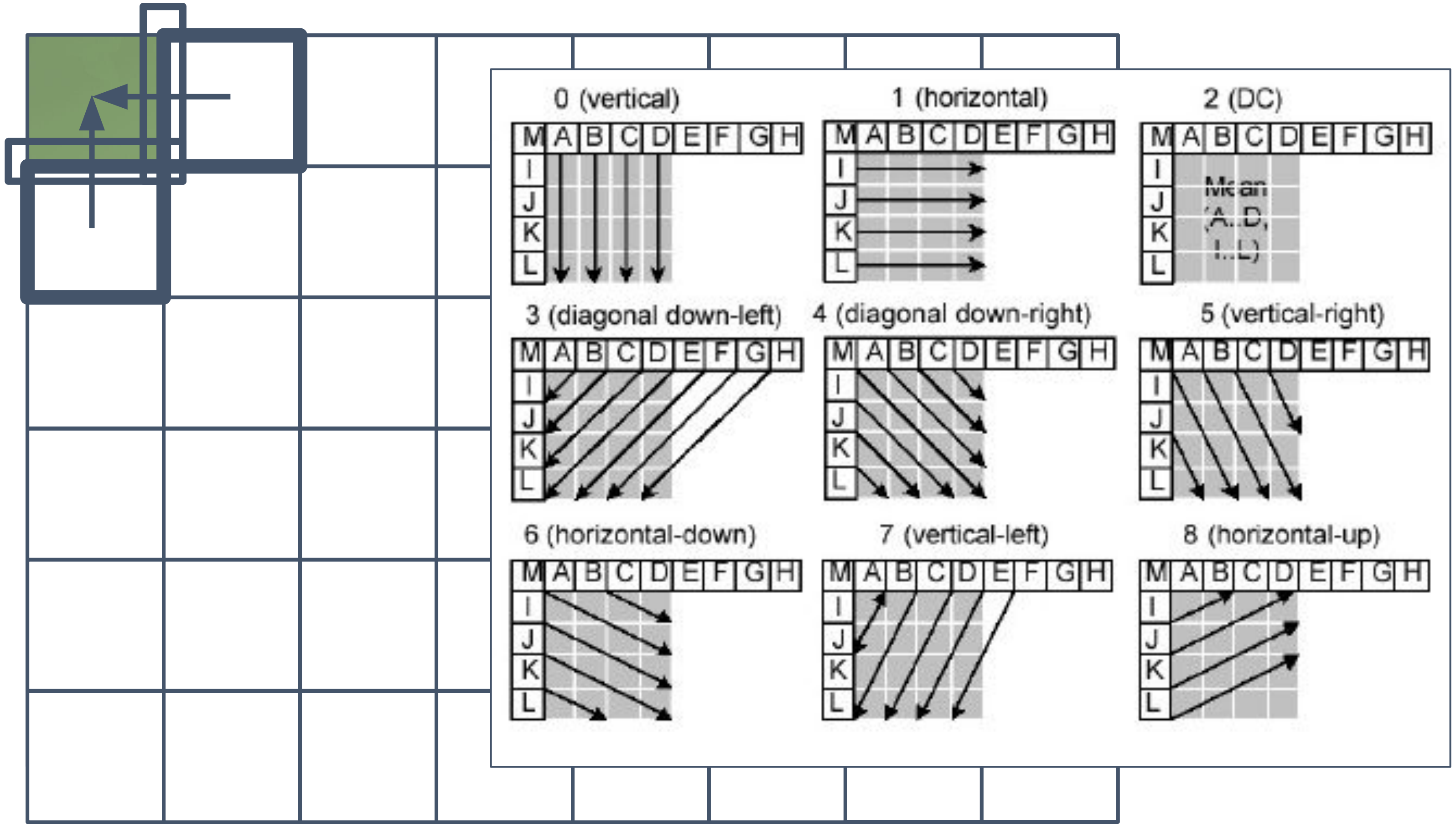
Intra Prediction

Send the first macroblock, then predict neighboring macroblocks



Intra Prediction

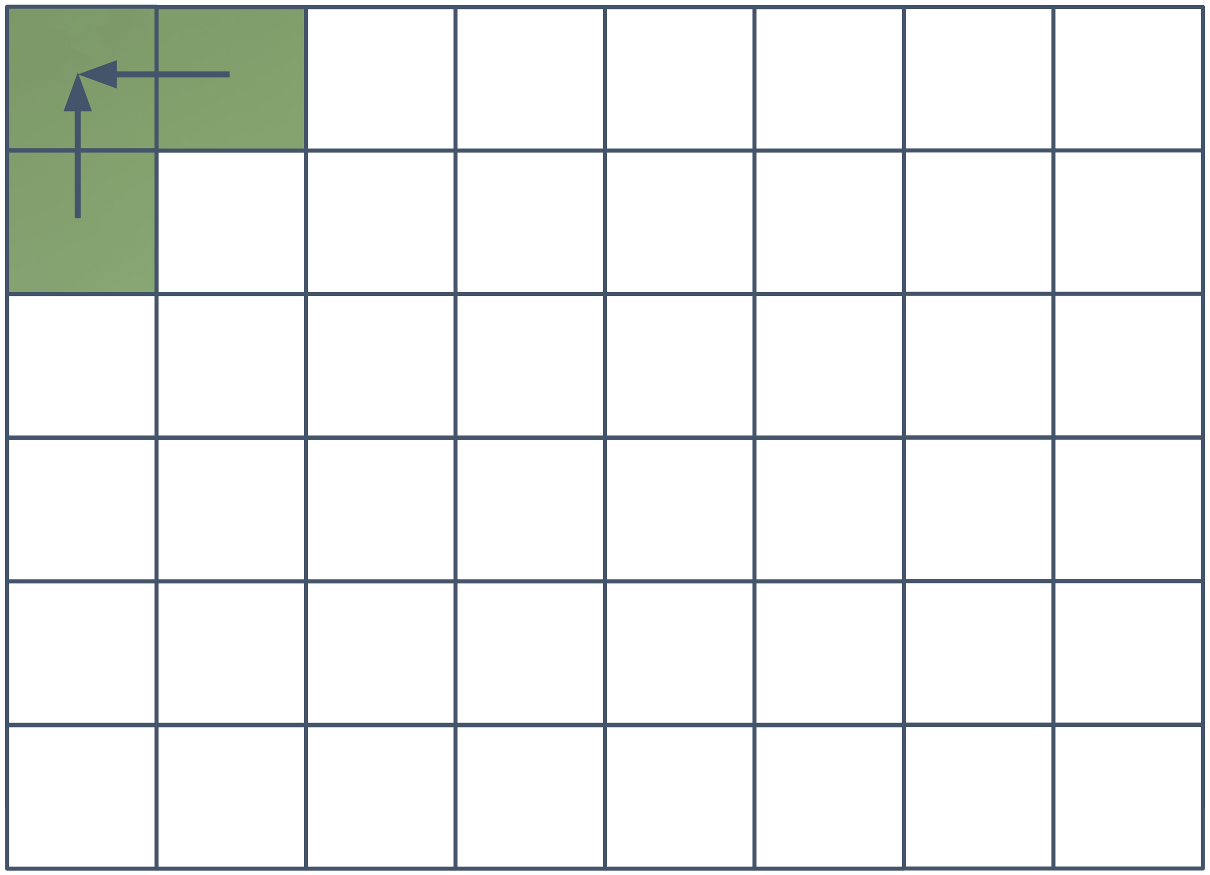
Send the first macroblock, then predict neighboring macroblocks



https://www.researchgate.net/publication/261700797/figure/fig3/AS:296664200957959@144774166551 8/The-nine-intra-prediction-modes-of-the-H264-standard-24-increased-in-HEVC-to-the-33.png

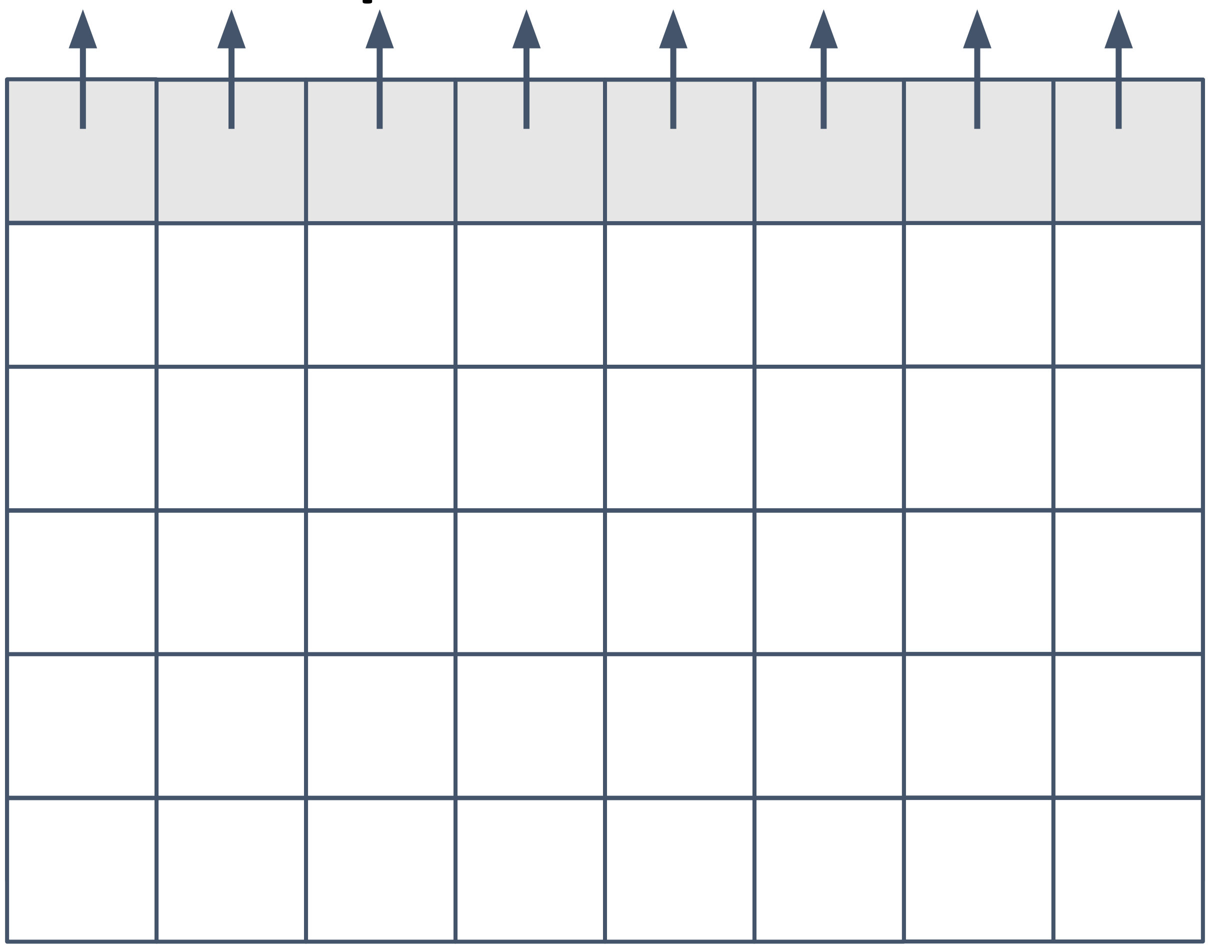
Intra Prediction

Send the first macroblock, then predict neighboring macroblocks

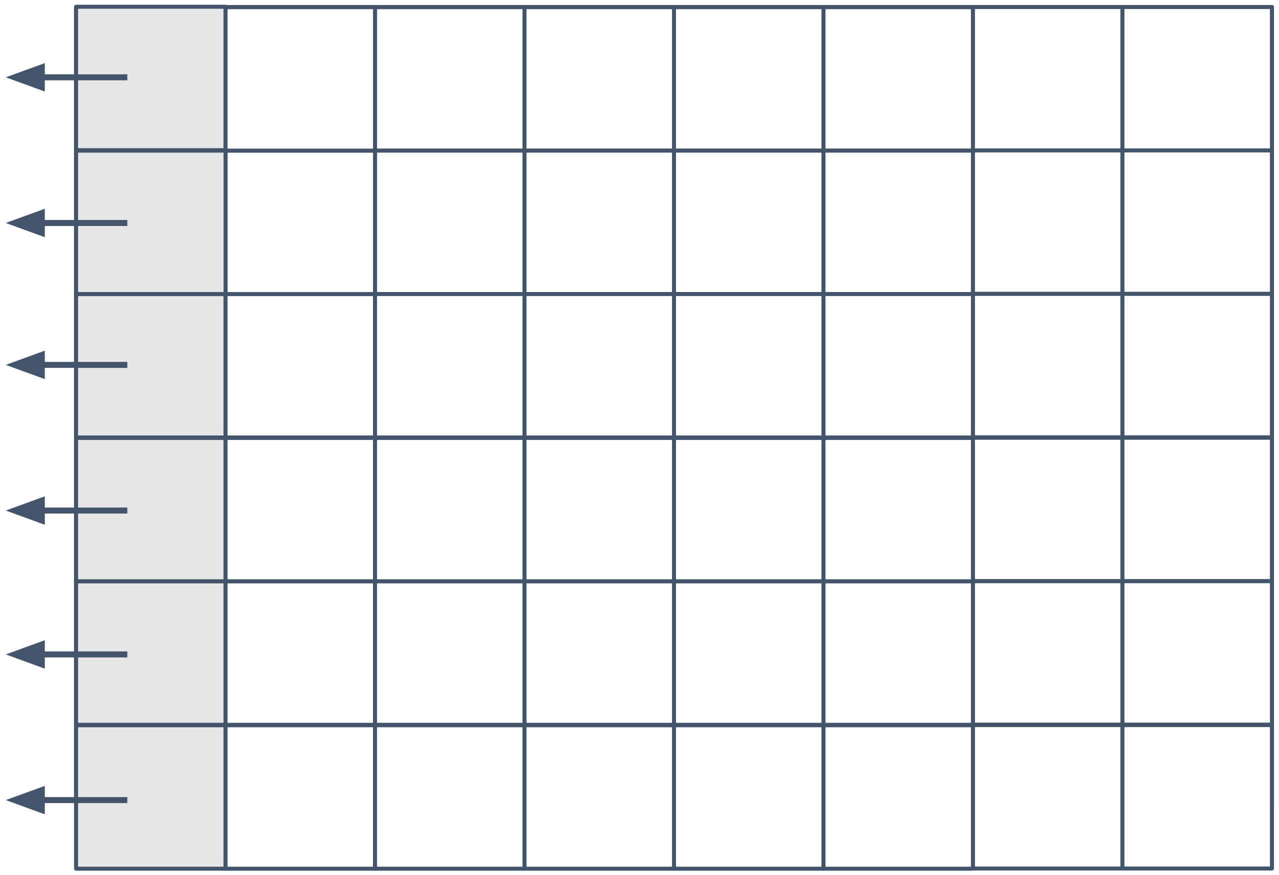


Prediction + Residue

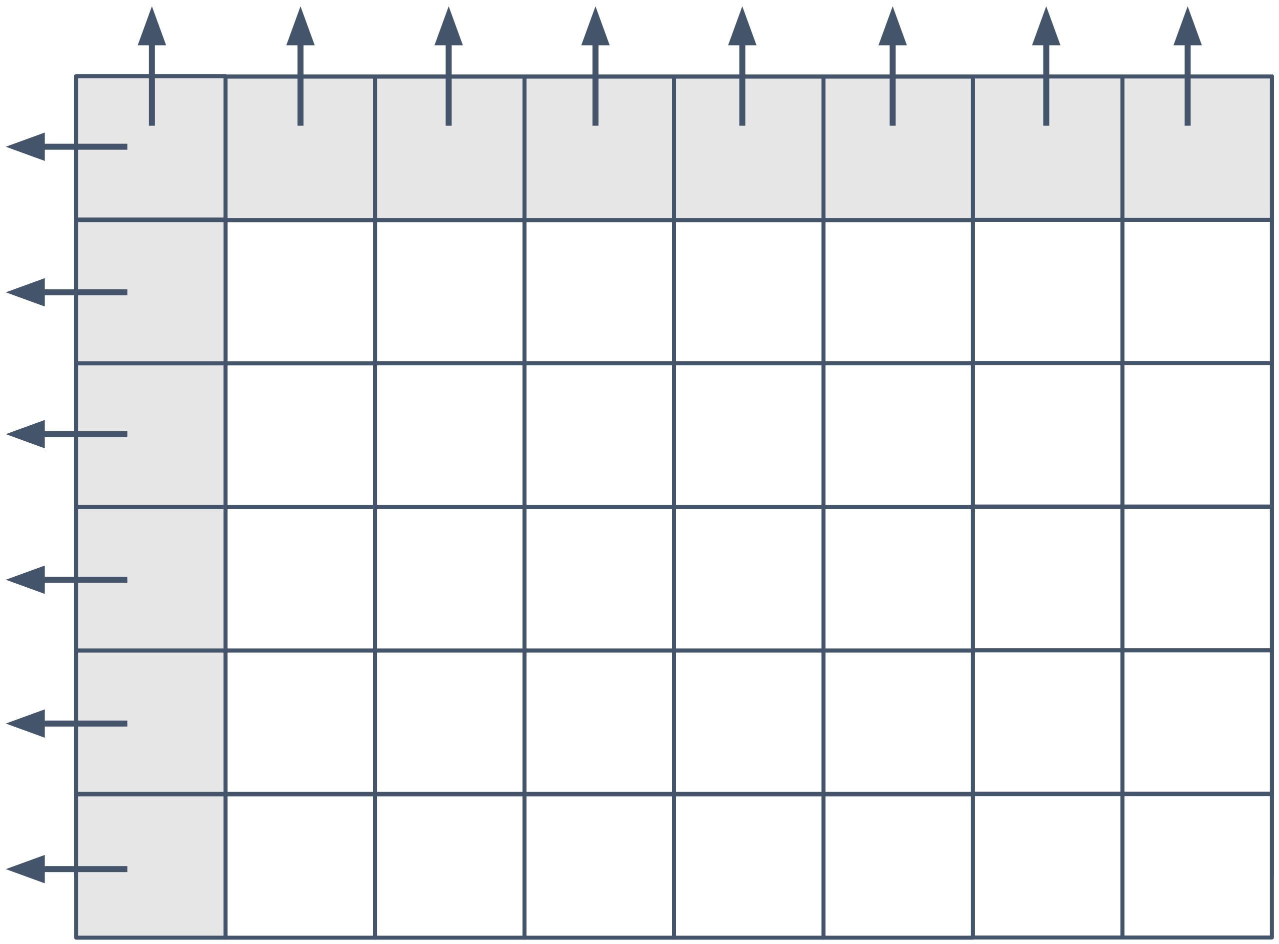
What if we tried to predict from above on the top-most row?



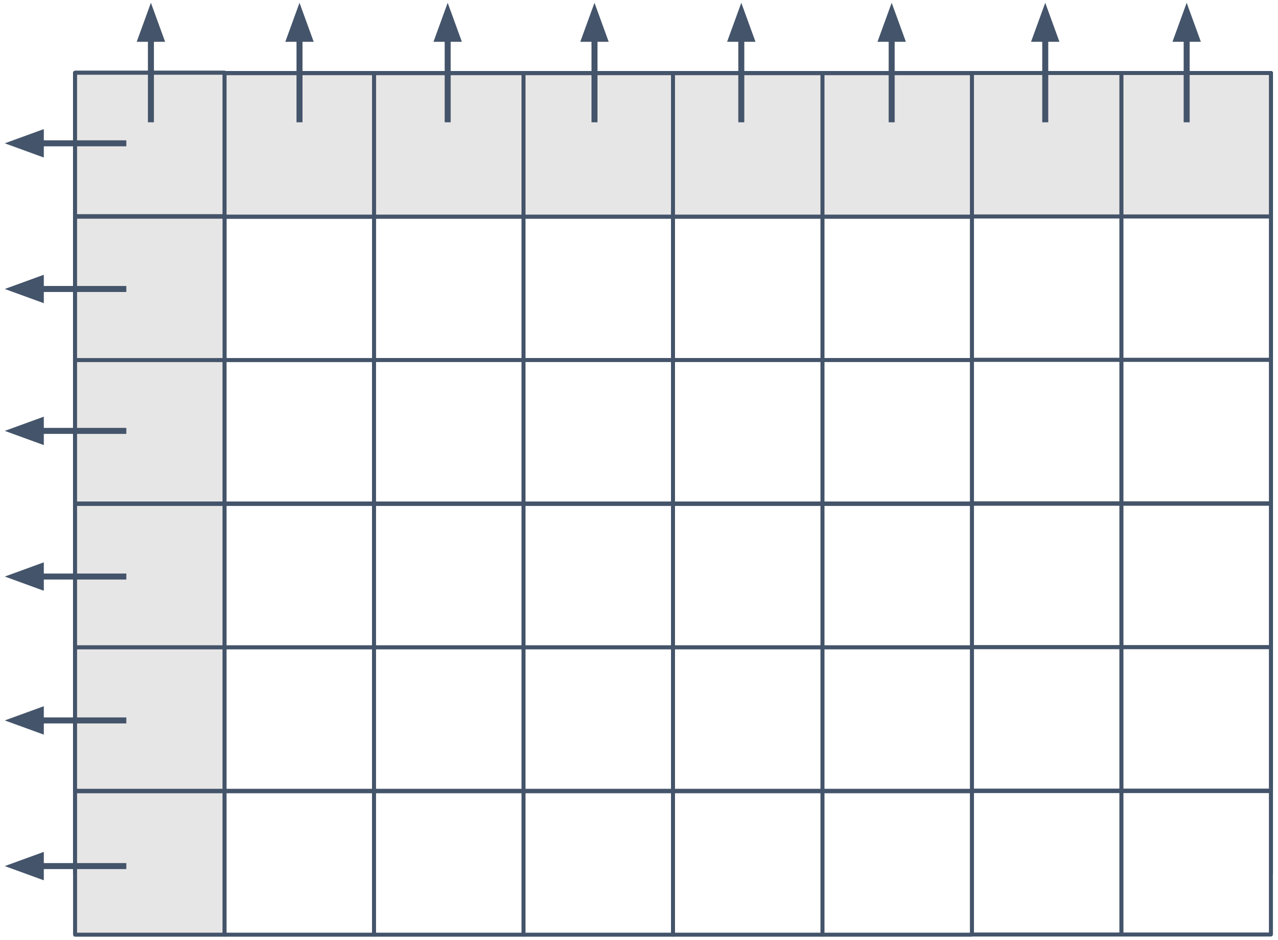
Or from the left on the left-most row?



Out-of-bounds Intra prediction

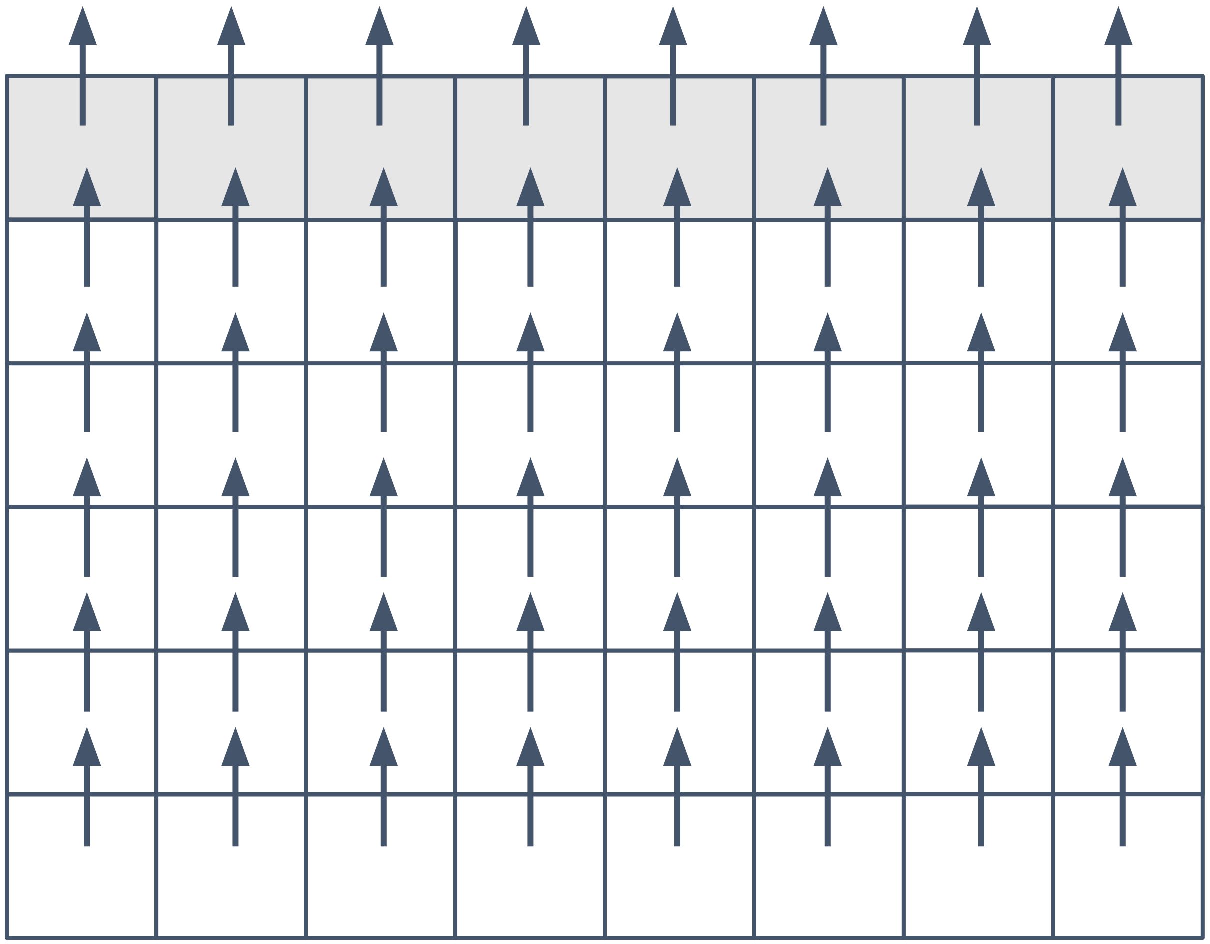


Found this leads to different results across hardware



Found this leads to different results across hardware

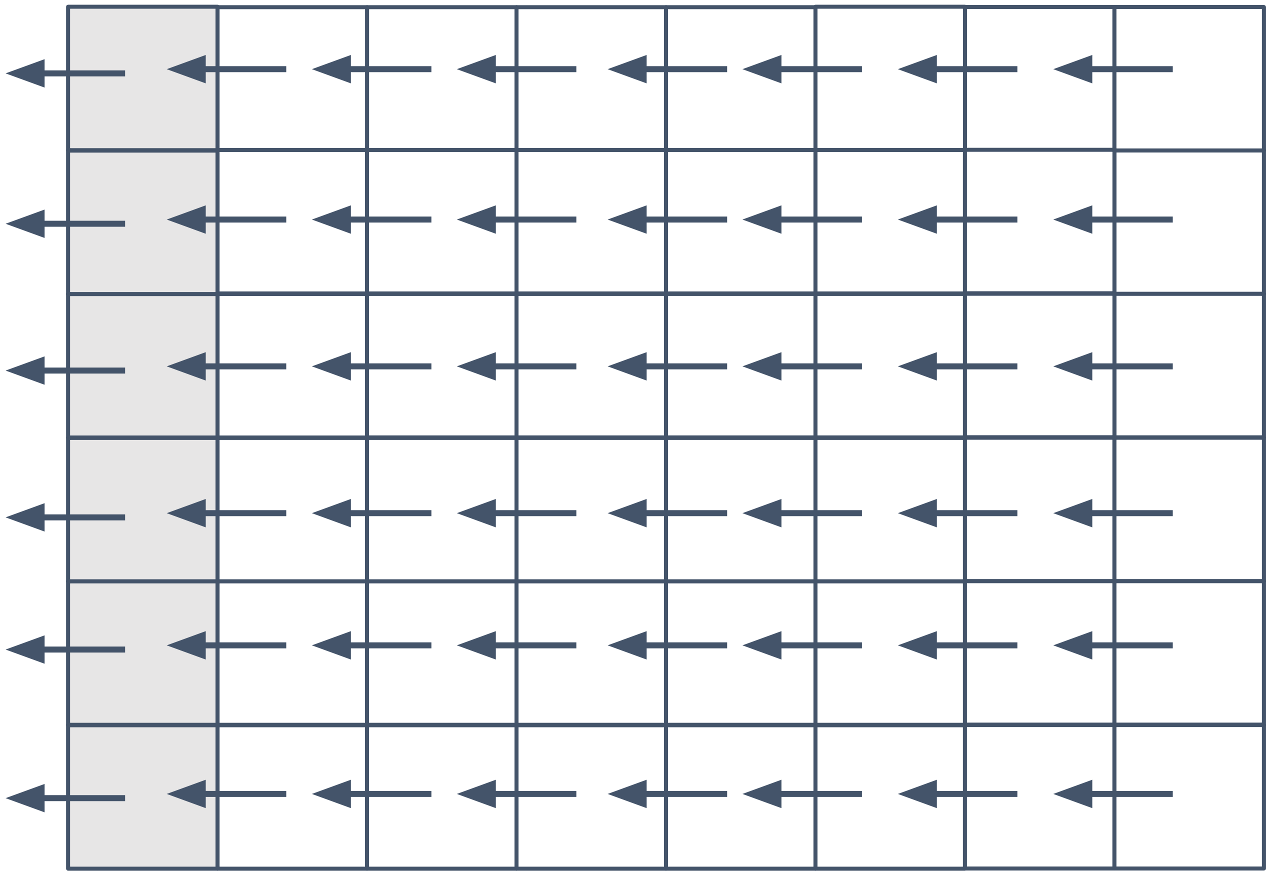
Amplified **Vertical** Prediction



No residue

Found this leads to different results across hardware

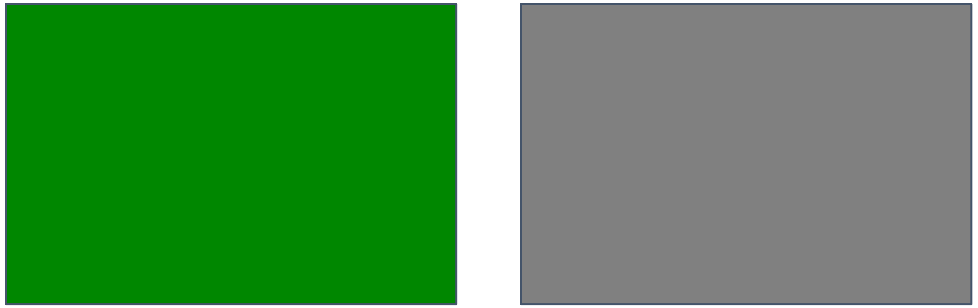
Amplified **Horizontal** Prediction



No residue

Results of out-of-bounds Intra Prediction

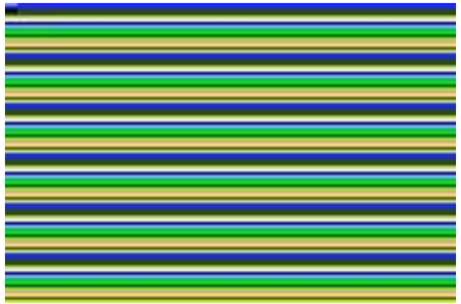
- Solid color (YUV all 0x00 or 0x80)



Results of out-of-bounds Intra Prediction

- Solid color (YUV all 0x00 or 0x80)
- Uninitialized data

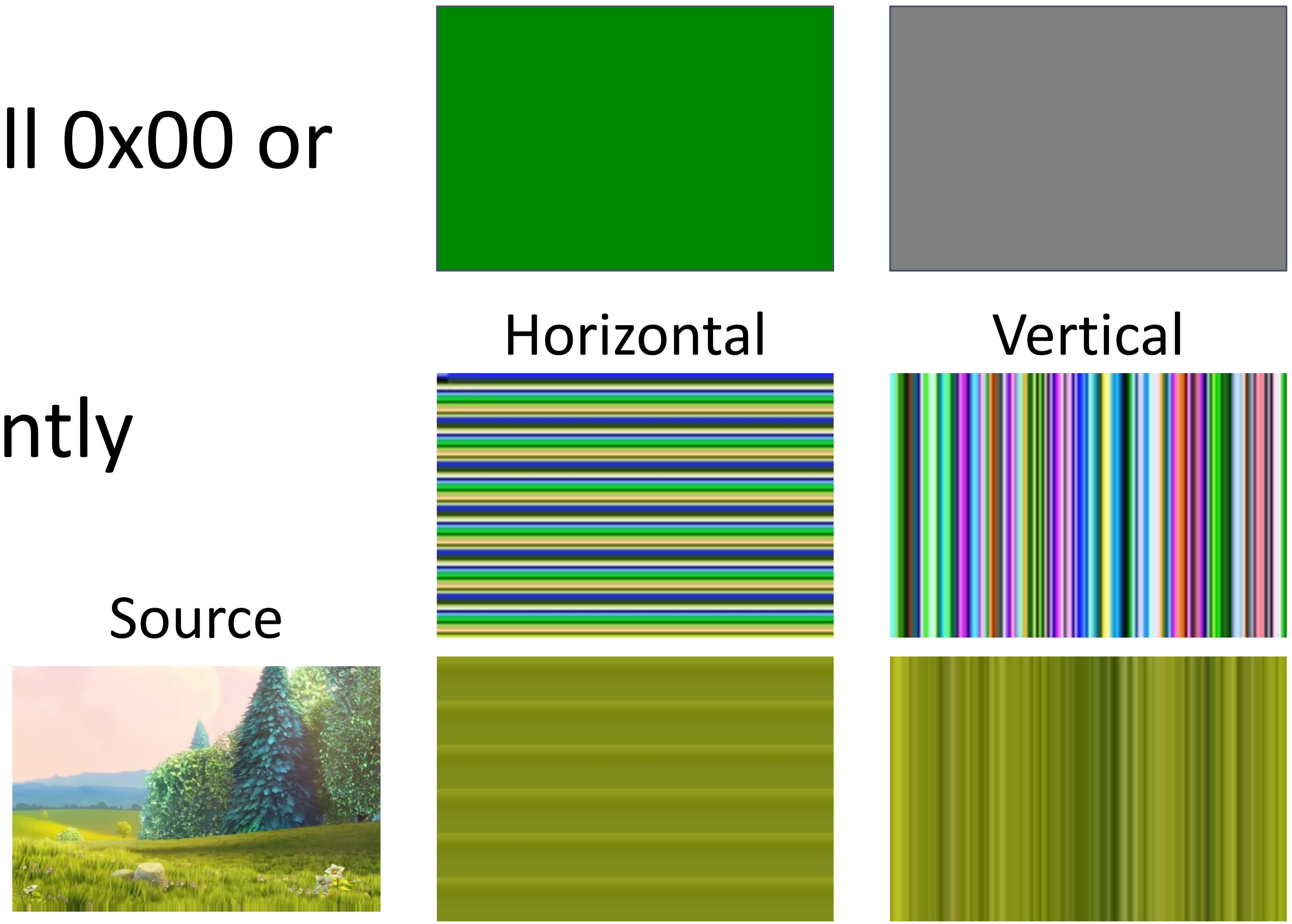
Horizontal



Vertical

Results of out-of-bounds Intra Prediction

- Solid color (YUV all 0x00 or 0x80)
- Uninitialized data
- Pixels from a recently decoded video

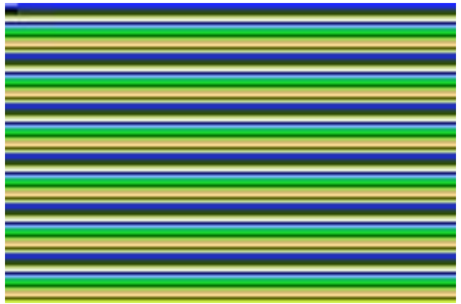


Results of out-of-bounds Intra Prediction

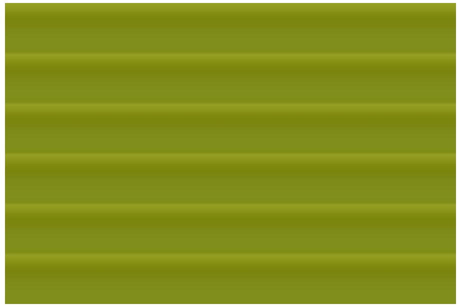
- Solid color (YUV all 0x00 or 0x80)
- Uninitialized data
- Pixels from a recently decoded video

Horizontal

Vertical



Source



Luma/Chroma Thief

Generating `Luma/Chroma Thief`

- No sane encoder would generate this

Generating **`Luma/Chroma Thief`**

- No sane encoder would generate this
- CABAC entropy-encoded syntax elements difficult to do by hand

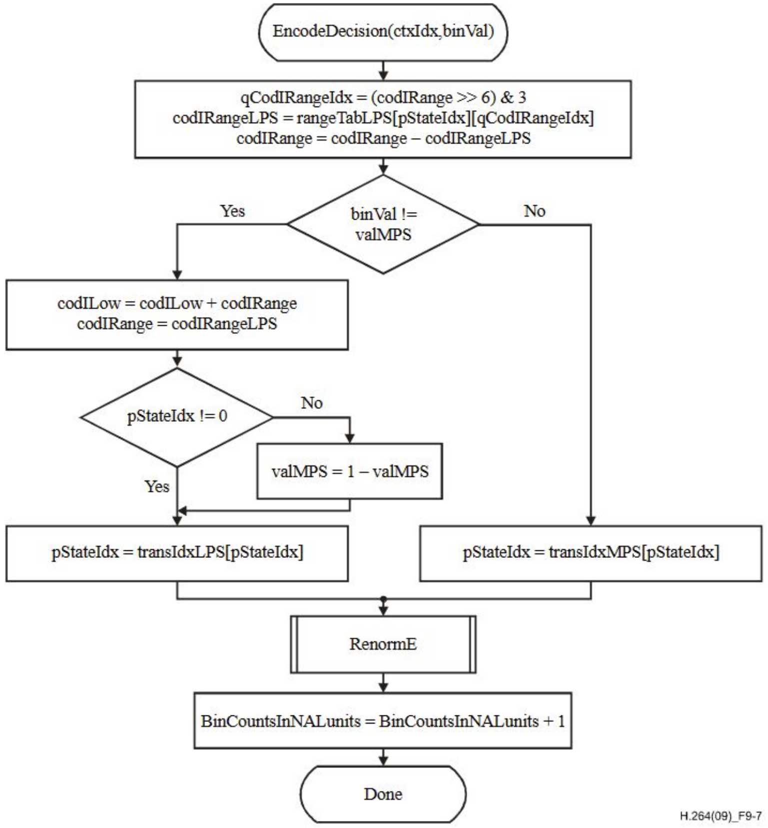


Figure 9-7 – Flowchart for encoding a decision

Generating `Luma/Chroma Thief`

- No sane encoder would generate this
- CABAC entropy-encoded syntax elements difficult to do by hand

H26Forge to the rescue!

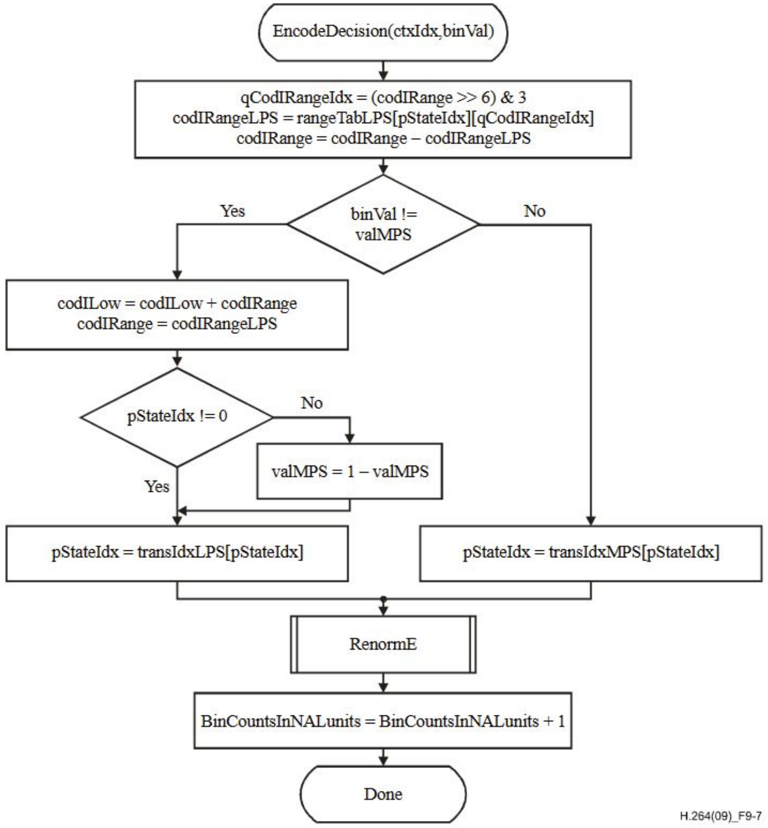


Figure 9-7 – Flowchart for encoding a decision

H26Forge **`Luma/Chroma Thief`** Transform

```python
# disable deblocking filter to remove stolen value post-processing
ds["ppses"][0]["deblocking_filter_control_present_flag"] = True
ds["slices"][0]["sh"]["disable_deblocking_filter_idc"] = 1

for i in range(len(ds["slices"][0]["sd"]["macroblock_vec"])):
    # 16x16 vertical luma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["mb_type"] = "I16x16_0_0_0"
    # make sure these values are correct for re-encoding
    ds["slices"][0]["sd"]["macroblock_vec"][i]["coded_block_pattern"] = 0
    ds = set_cbp_chroma_and_luma(0, i, ds)
    # vertical intra chroma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["intra_chroma_pred_mode"] = 2
```

H26Forge **Luma/Chroma Thief** Transform

```python
# disable deblocking filter to remove stolen value post-processing
ds["ppses"][0]["deblocking_filter_control_present_flag"] = True
ds["slices"][0]["sh"]["disable_deblocking_filter_idc"] = 1

for i in range(len(ds["slices"][0]["sd"]["macroblock_vec"])):
    # 16x16 vertical luma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["mb_type"] = "I16x16_0_0_0"
    # make sure these values are correct for re-encoding
    ds["slices"][0]["sd"]["macroblock_vec"][i]["coded_block_pattern"] = 0
    ds = set_cbp_chroma_and_luma(0, i, ds)
    # vertical intra chroma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["intra_chroma_pred_mode"] = 2
```

H26Forge **Luma/Chroma Thief** Transform

```
# disable deblocking filter to remove stolen value post-processing
ds["ppses"][0]["deblocking_filter_control_present_flag"] = True
ds["slices"][0]["sh"]["disable_deblocking_filter_idc"] = 1

for i in range(len(ds["slices"][0]["sd"]["macroblock_vec"])):
    # 16x16 vertical luma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["mb_type"] = "I16x16_0_0_0"
    # make sure these values are correct for re-encoding
    ds["slices"][0]["sd"]["macroblock_vec"][i]["coded_block_pattern"] = 0
    ds = set_cbp_chroma_and_luma(0, i, ds)
    # vertical intra chroma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["intra_chroma_pred_mode"] = 2
```

H26Forge **Luma/Chroma Thief** Transform

```python
# disable deblocking filter to remove stolen value post-processing
ds["ppses"][0]["deblocking_filter_control_present_flag"] = True
ds["slices"][0]["sh"]["disable_deblocking_filter_idc"] = 1

for i in range(len(ds["slices"][0]["sd"]["macroblock_vec"])):
    # 16x16 vertical luma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["mb_type"] = "I16x16_0_0_0"
    # make sure these values are correct for re-encoding
    ds["slices"][0]["sd"]["macroblock_vec"][i]["coded_block_pattern"] = 0
    ds = set_cbp_chroma_and_luma(0, i, ds)
    # vertical intra chroma prediction
    ds["slices"][0]["sd"]["macroblock_vec"][i]["intra_chroma_pred_mode"] = 2
```

Vulnerability Hunting with H26Forge

Can use H26Forge to generate syntactically correct but semantically non-compliant videos for fuzz testing

Can create proof-of-concept videos with H26Forge's video transform capability

Vulnerability Hunting with H26Forge

Can use H26Forge to generate syntactically correct but semantically non-compliant videos for fuzz testing

Can create proof-of-concept videos with H26Forge's video transform capability

You can start using H26Forge NOW to find issues before attackers do!
https://github.com/h26forge/h26forge

Datamoshing

Datamoshing

- Common techniques

Datamoshing

- Common techniques
  - I-Frame removal

https://i.kym-cdn.com/photos/images/original/000/475/532/d3d.gif

Datamoshing

- Common techniques
  - I-Frame removal
  - Inter frame copying

https://i.kym-cdn.com/photos/images/original/000/475/532/d3d.gif

https://i.kym-cdn.com/photos/images/original/000/606/989/31b.gif

Datamoshing

- Common techniques
  - I-Frame removal
  - Inter frame copying
- Existing tutorials rely on using hex editors or Avidemux

TUTORIALS, VIDEO

HOW TO DATAMOSH VIDEOS WITH DATA CORRUPTION

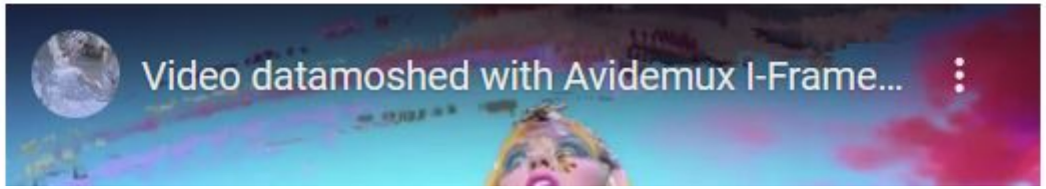
▶ VIDEO   JUNE 17, 2016   PHIL   5 COMMENTS

http://datamoshing.com/2016/06/17/how-to-datamosh-videos-with-data-corruption/

TUTORIALS, VIDEO

HOW TO DATAMOSH VIDEOS

▶ VIDEO   JUNE 26, 2016   PHIL   41 COMMENTS



http://datamoshing.com/2016/06/26/how-to-datamosh-videos/

Datamoshing

- Common techniques
  - I-Frame removal
  - Inter frame copying
- Existing tutorials rely on using hex editors or Avidemux

Datamoshing with H26Forge

TUTORIALS, VIDEO

HOW TO DATAMOSH VIDEOS WITH DATA CORRUPTION

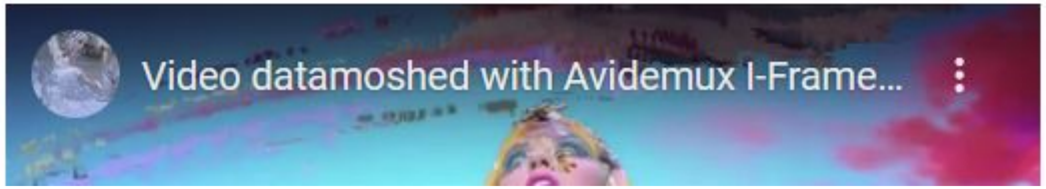
▶ VIDEO   JUNE 17, 2016   PHIL   5 COMMENTS

http://datamoshing.com/2016/06/17/how-to-datamosh-videos-with-data-corruption/

TUTORIALS, VIDEO

HOW TO DATAMOSH VIDEOS

▶ VIDEO   JUNE 26, 2016   PHIL   41 COMMENTS



http://datamoshing.com/2016/06/26/how-to-datamosh-videos/

Datamoshing with H26Forge

Datamoshing with H26Forge

Datamoshing with H26Forge

```
# duplicate P slices dup_amount times
if is_slice_type(ds["slices"][slice_idx]["sh"]["slice_type"], 'P'):
    for _ in range(dup_amount):
        slice_idx += 1
        # copy over header elements
        ds["nalu_headers"].insert(i, ds["nalu_headers"][i])
        ds["nalu_elements"].insert(i, ds["nalu_elements"][i])
        # copy over slices
        ds["slices"].insert(slice_idx, copy.deepcopy(ds["slices"][og_slice_idx]))
        ds["slices"][slice_idx]["sh"]["frame_num"] = frame_num
        ds["slices"][slice_idx]["sh"]["pic_order_cnt_lsb"] = pic_order_cnt_lsb
        frame_num += 1
        pic_order_cnt_lsb += 2
    i += dup_amount
```

Datamoshing with H26Forge

```
# duplicate P slices dup_amount times
if is_slice_type(ds["slices"][slice_idx]["sh"]["slice_type"], 'P'):
    for _ in range(dup_amount):
        slice_idx += 1
        # copy over header elements
        ds["nalu_headers"].insert(i, ds["nalu_headers"][i])
        ds["nalu_elements"].insert(i, ds["nalu_elements"][i])
        # copy over slices
        ds["slices"].insert(slice_idx, copy.deepcopy(ds["slices"][og_slice_idx]))
        ds["slices"][slice_idx]["sh"]["frame_num"] = frame_num
        ds["slices"][slice_idx]["sh"]["pic_order_cnt_lsb"] = pic_order_cnt_lsb
        frame_num += 1
        pic_order_cnt_lsb += 2
    i += dup_amount
```

Datamoshing with H26Forge

```
# duplicate P slices dup_amount times
if is_slice_type(ds["slices"][slice_idx]["sh"]["slice_type"], 'P'):
    for _ in range(dup_amount):
        slice_idx += 1
```

```
ents
rt(i, ds["nalu_headers"][i])
ert(i, ds["nalu_elements"][i])

ce_idx, copy.deepcopy(ds["slices"][og_slice_idx]))
["sh"]["frame_num"] = frame_num
["sh"]["pic_order_cnt_lsb"] = pic_order_cnt_lsb
```

Datamoshing with H26Forge

```python
# duplicate P slices dup_amount times
if is_slice_type(ds["slices"][slice_idx]["sh"]["slice_type"], 'P'):
    for _ in range(dup_amount):
        slice_idx += 1
```

More Use Cases for H26Forge

More Use Cases for H26Forge

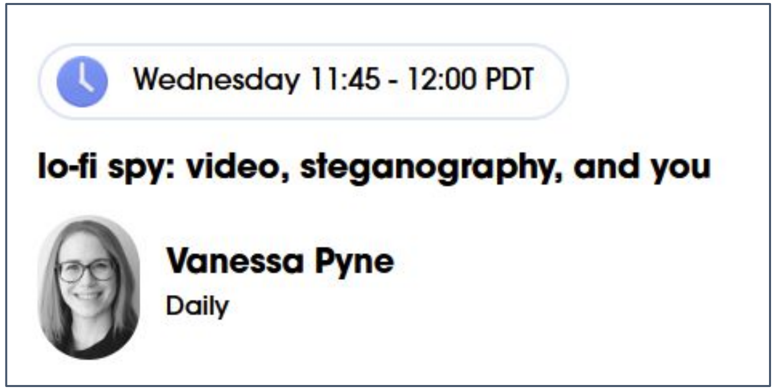
1. **Syntax element surgery**: identifying and fixing issues in transmitted videos

https://cdn11.bigcommerce.com/s-ufhcuzfxw9/images/stencil/1280x1280/products/12888/16148/DE-SCALP22__61649.1503517947.jpg?c=2

More Use Cases for H26Forge

1. **Syntax element surgery**: identifying and fixing issues in transmitted videos
2. **Steganography**: hiding messages in syntax element values

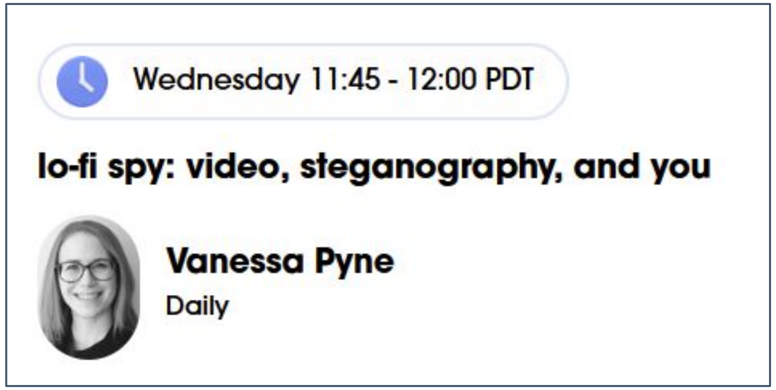
https://cdn11.bigcommerce.com/s-ufhcuzfxw9/images/stencil/1280x1280/products/12888/16148/DE-SCALP22__61649.1503517947.jpg?c=2



More Use Cases for H26Forge

1. **Syntax element surgery**: identifying and fixing issues in transmitted videos
2. **Steganography**: hiding messages in syntax element values
3. **In-the-wild exploit detection**: monitoring for anomalous syntax elements

https://cdn11.bigcommerce.com/s-ufhcuzfxw9/images/stencil/1280x1280/products/12888/16148/DE-SCALP22__61649.1503517947.jpg?c=2



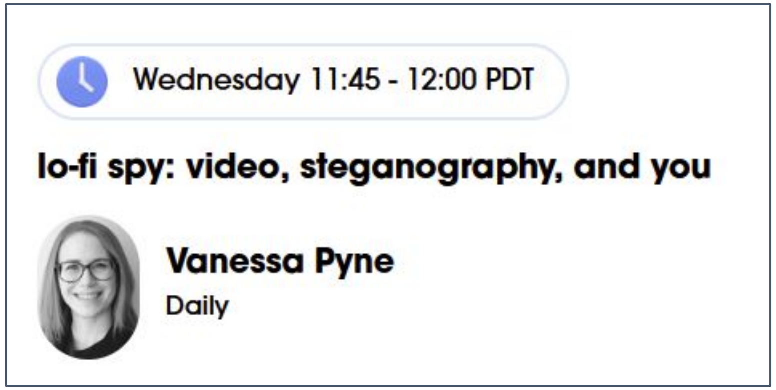
0-days In-the-Wild

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

More Use Cases for H26Forge

1. **Syntax element surgery**: identifying and fixing issues in transmitted videos
2. **Steganography**: hiding messages in syntax element values
3. **In-the-wild exploit detection**: monitoring for anomalous syntax elements
4. ???

https://cdn11.bigcommerce.com/s-ufhcuzfxw9/images/stencil/1280x1280/products/12888/16148/DE-SCALP22__61649.1503517947.jpg?c=2



0-days In-the-Wild

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

Conclusion

H26Forge provides the capability to programmatically modify syntax elements, allowing for vulnerability hunting, datamoshing, and potentially more new applications.

Conclusion

H26Forge provides the capability to programmatically modify syntax elements, allowing for vulnerability hunting, datamoshing, and potentially more new applications.

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.

https://github.com/h26forge/h26forge

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 pro-

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

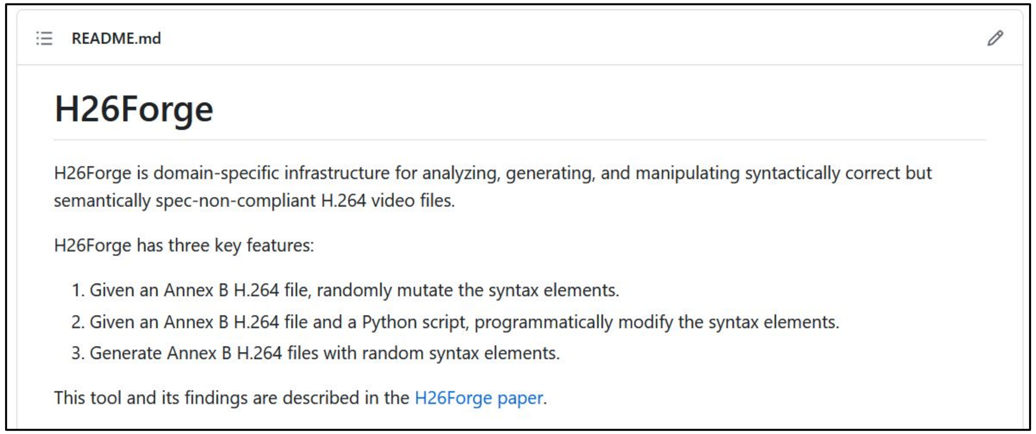
On the basis of a guideline they call "The Rule Of 2,"[2] the Chrome developers try to avoid writing code that does more

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

Conclusion

H26Forge provides the capability to programmatically modify syntax elements, allowing for vulnerability hunting, datamoshing, and potentially more new applications.

Contributions and feature requests welcome!



README.md

H26Forge

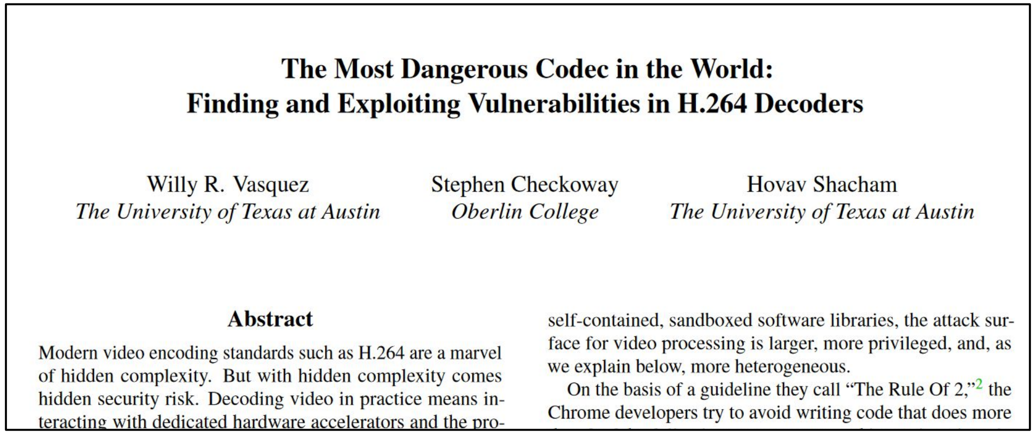
H26Forge is domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant H.264 video files.

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.

https://github.com/h26forge/h26forge



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 pro-

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

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

Conclusion

H26Forge provides the capability to programmatically modify syntax elements, allowing for vulnerability hunting, datamoshing, and potentially more new applications.

Contributions and feature requests welcome!

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.

https://github.com/h26forge/h26forge

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 pro-

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

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