HIGH-DEF FUZZING

EXPLOITATION OVER HDMI-CEC


```
name = "Joshua Smith"
job  = "Senior Security Researcher"
job += "Zero Day Initiative"
irc  = "kernelsmith"
twit = "@kernelsmith"
```


ZERO DAY
INITIATIVE

Previous Research

- HDMI – Hacking Displays Made Interesting
  - Andy Davis
  - BlackHat EU 2012

What is HDMI?

High Def Multimedia Interface

- HDMI is a specification
- Implemented as Cables & Connectors
- Successor to DVI
- Has Quite a Few Features

What is CEC?

Consumer Electronics Control

- HDMI feature
- Allows user to command & control up to 15 devices
- Can relay commands from remotes
- It is what automatically changes your TV input
- Has some other intriguing features...

Why?

- Wanted to research an area that was relatively untouched
- I do not have mad hardware skills
- I like RISC targets & assembly
- Another attack vector for mobile devices via:
  - Mobile High-Definition Link (MHL) ~ Samsung & HTC
  - Slimport ~ LG, Google Nexus, Blackberry
- My son is completely obsessed with cords/wires, esp HDMI

Specs & Features

History

- 1.0 (Dec 2002), 1.1 (May 2004), 1.2 (Aug 2005)
  - Boring stuff
- 1.2a (Dec 2005)
  - Fully specified Consumer Electronics Control
  - This is the **good** stuff, for vulnerabilities anyway

Specs & Features

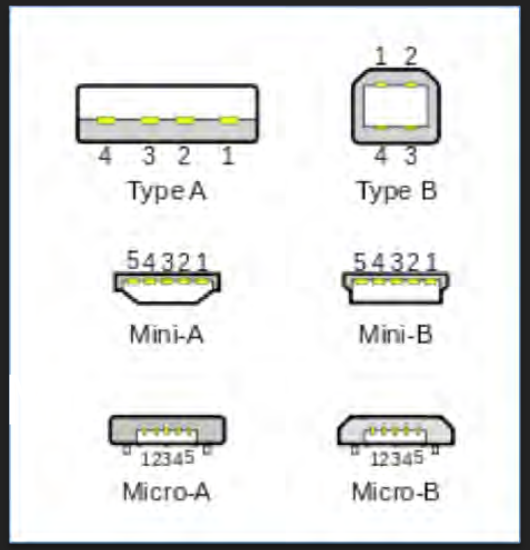
History Continued

- 1.3 - 1.3c (Jun 2006 through Aug 2008)
  - Whizz-bang A/V improvements & new connectors
- 1.4 (May 2009)
  - **Most widely** deployed and available
  - Features++: 4k, HEC, ARC, 3D, micro connector
  - Some that might interest us (next)
- 2.0 (Sep 2013)
  - New hotness: 4K video @60fps, Dual View, 3D++, CEC++

Specs & Features

Interesting 1.4 Features

- HEC (HDMI Ethernet Connection)
  - Sounds tasty
  - 100Mb/s
  - Enables traditional networking w/HD
- ARC (Audio Return Channel)



CEC Details

- 1-wire bidirectional serial bus
- Slow: 500Mb/s
- Uses AV.link protocol to perform remote control functions
- For HDMI:
  - CEC wiring is mandatory
  - CEC functionality (software mainly) is **optional**

CEC's Goals

- Simplify system integration
- Common protocol
- Extendable (vendor-specific commands)
- Commands are grouped together into Feature Sets
  - For example, one-touch play (OTP)
    - TV on, text view on (optional), set active source

Notable Implementations

- Commercial industry uses various trade names
  - Anynet+ (Samsung), Aquos Link (Sharp), BRAVIA Link/Sync (Sony)
  - SimpLink (LG), VIERA Link (Panasonic), EasyLink (Philips), etc
- Open Source
  - libCEC (dual commercial license)
  - Android HDMI-CEC

Not-HDMI CEC

- Slimport
- Mobile High-Definition Link (MHL)

Notes: TODO: add tidbits about Slimport and MHL, like overloading the connector etc

CEC Addressing

PHYSICAL

- N.N.N.N where 0x0<=N<=0xF
- Like F.A.4.0
- Obtained on hot-plug from EDID
- The root display is always 0.0.0.0
- If attached to 1st input on root: 1.0.0.0
- Required as CEC has a notion of switching

CEC Addressing

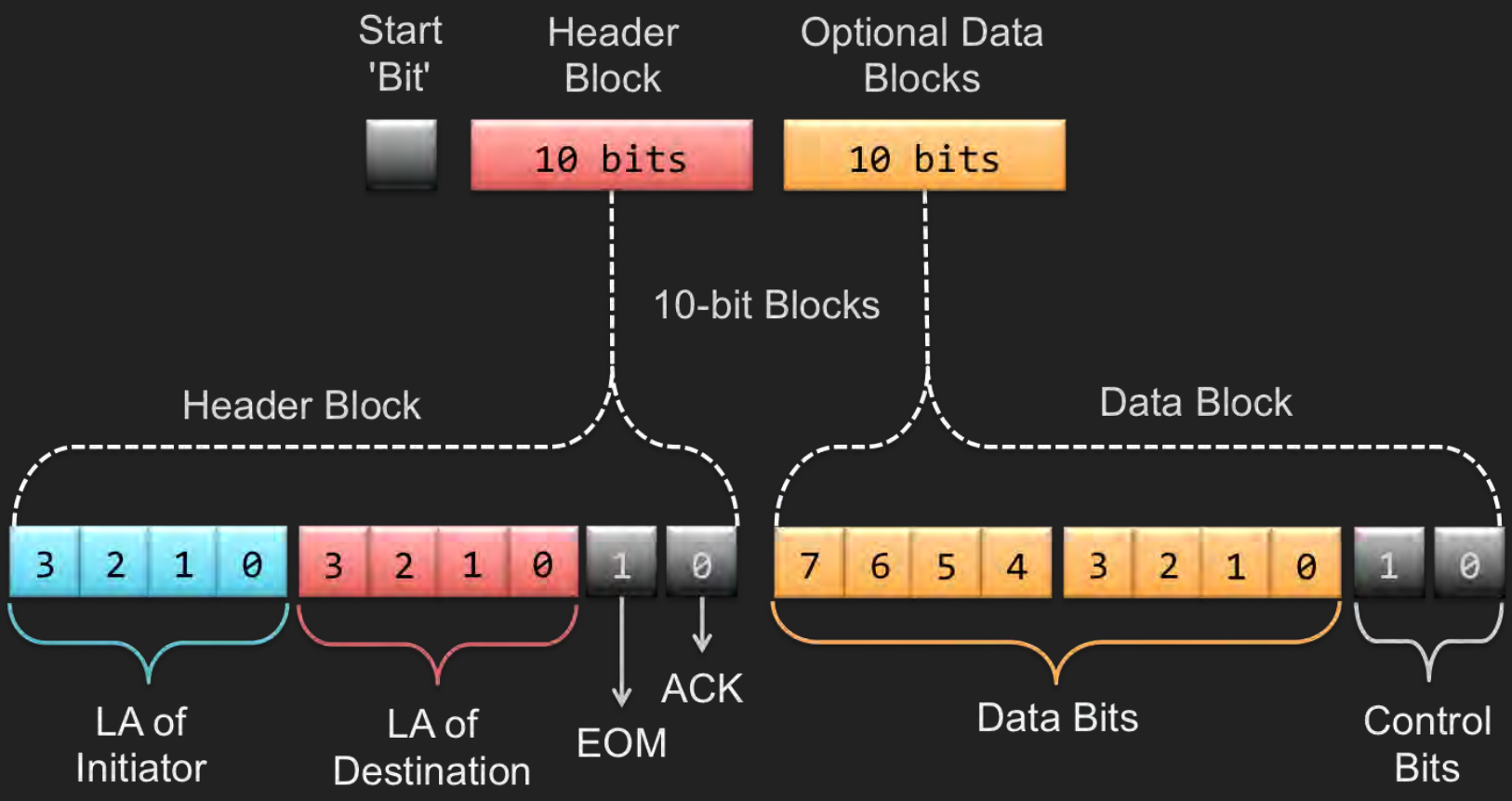
LOGICAL

- L where 0x0<=L<=0xF
- Root display is always 0
- By product type
- Negotiated w/other devices
- Example: first STB in system is always 3
- Non-CEC devices only have physical addr

Logical Addresses

Address	Device	Address	Device
0	TV	8	Playback Dev 2
1	Rec. Device 1	9	Rec Device 3
2	Rec. Device 2	10	Tuner 4
3	Tuner 1	11	Playback Dev 3
4	Playback Dev 1	12	Reserved
5	Audio System	13	Reserved
6	Tuner 2	14	Free Use
7	Tuner 3	15	Unreg/Broadcast

CEC Protocol



Blocks & Frames

- Blocks
  - Each block is 10 bits
  - Max of 16 blocks (14 purely data blocks)
- Frames
  - (1bit) Start bit
  - (10bits) Header block
  - (10bits) Opcode block
  - (10bits) Optional data block(s)

Header Block

Source	Dest	EoM	Ack
3 2 1 0	3 2 1 0	E	A

- (4bits) Logical address of source
- (4bits) Logical address of dest
- (2bits) Control bits (EoM & Ack)
- Example: 0100:0000:0:0 = Src 4, Dest 0

Data Block

Data	EoM	Ack
7 6 5 4 3 2 1 0	E	A

- (8bits) Data (Big-endian/MSB first)
- (2bits) Control bits (EoM & Ack)
- Example: 01000001:1:0 = "A"

CEC Protocol

Pinging and Polling

- The "Ping"
  - EOM bit in header is set to 1
  - Used to poll for devices etc (fuzz monitor?)
    - Source & dest addresses will be different
  - Also used for allocating Logical Addresses
    - Source & dest addresses are the same

CEC Protocol

Additional Info

- All numbers > 1 byte are transmitted as big-endian
- All bit sequences are sent MSB first
- Messages can be directly addressed, broadcast, or both
- Should ignore a message coming from address 15, unless:
  - Message invokes a broadcast response
  - Message has been sent by a CEC Switch
  - The message is **Standby**

CEC Protocol

The Long and Short of It...

- 10:64:44:65:66:43:6F:6E:20:32:33
- 1F:82:10.00
- SD:OP:41:42:43:44:45:46

CEC Protocol

Example Messages

Name	ID	Feature Set	Addr	Parameters
Poll		Sys Info	Direct	
Get CEC Ver	9F	Sys Info	Direct	
CEC Version	9E	Sys Info	Direct	CEC Version
Set OSD Name	47	OSD Xfer	Direct	OSD Name
Set OSD Str	64	OSD Disp	Direct	DispCtrl,Str
Active Source	82	OTP, RC	Bcast	Phys Addr

CEC Protocol

Transmission (Flow) Control

- 3 mechanisms to provide reliable frame transfer
  1. Frame re-transmissions (1 to 5)
  2. Flow control
  3. Frame validation (ignore msgs w/wrong #args)
- A message is assumed correctly received when:
  - It has been transmitted and acknowledged
- A message is assumed to have been acted upon when:
  - Sender does not receive `Feature Abort` w/in 1sec

Common Sequences

- Addressing
  1. Discovery (poll etc) of new physical address
  2. Allocation (of logical address)
  3. Report by broadcasting `ReportPhysicalAddress`
- Become active source
  1. Broadcast an `ActiveSource` to declare intention
  2. Presently active source shall act appropriately

Feature Sets

One-Touch Play (OTP)

- ImageViewOn* 40:04 (assumes playback dev 1)
- TextViewOn 4F:0D (optional, remove displayed menus)
- ActiveSource 4F:82 (assumes playback dev 1)

Attack Vectors

- HDMI Ethernet Channel (HEC)
- Network connectivity to things thought un-networked
- Great place to hide
- Targetable devices
  - TVs, BluRays, receivers, "TV Sticks", game consoles?
  - Mobile phones & tablets
    - Devices implementing MHL/Slimport
    - Known popular mobile devices that implement MHL

Attack Surface

- CEC commands
- HEC commands
- CDC commands

Finding Vulns

Approaches

- Identify "at-risk" messages & fuzz
- Source Code Analysis
  - Hard to come by except libCEC & Android
- Reverse Engineering
  - Can be hard to get all the firmwarez
- Expect different architectures
  - MIPS, ARM, ARC etc
  - MIPS is generally most popular so far

Interesting Messages

- String operations
  - Set OSD Name (0x47)
    - Preferred name for use in any OSD (menus)
  - Set OSD String (0x64)
    - Text string to the TV for display
  - Set Timer Program Title (0x67)
    - Set the name of a program associated w/a timer
  - Vendor-specific Messages
    - Because who knows what they might do

In Order to Fuzz

We Need to Answer Some Questions

- How can we send arbitrary CEC messages?
- How can we detect if a crash occurred?

Sending Messages

Hardware

- ~0 {lap,desk}tops with HDMI-CEC
  - Many have HDMI, none have CEC
- Adapters
  - Pulse-Eight USB-HDMI
  - RainShadow HDMI-CEC to USB Brid
- Raspberry Pi
- RPi & P8 adapter both use libCEC :)

Sending Messages

Software

- Pulse-Eight driver is open source (libCEC)
  - Dual-licensed actually (GPLv2/Commercial)
  - Python SWIG-based bindings
  - Supports a handful of devices

Fuzzing CEC

libCEC

- Can send CEC messages with:
  - Raspberry Pi + libCEC
  - P8 USB-HDMI adapter + libCEC
- But can we really send arbitrary CEC messages?

```
lib.Transmit(CommandFromString("10:82:41:41:41:41:41:41:41:41:41"))
```

YES. It would appear at least.

To know for sure, had to ensure libCEC was not validating.

Demo

Fuzzing Process

- It has been done with Python + RainbowTech serial API
  - I actually did not know this until late in the research
  - RainbowTech device has a nice simple serial API
  - Not much complex functionality
  - I had already started down the path below
- libCEC + Python since pyCecClient is already a thing
  - Can use the P8 USB adapter and/or Raspberry Pi(s)
  - May port to Ruby since SWIG & Ruby++

Fuzzing Process

Major Steps

- ID Target and Inputs
- Generate Fuzzed Data
- Execute Fuzzed Data
- Monitor for Exceptions
- Determine Exploitability

Fuzzing: Brute Force Vulnerability Discovery (Sutton, Michael; Greene, Adam; Amini, Pedram)

Generate Fuzzed Data

- Started with "long" strings and string-based messages
- Format strings
- Parameter abuse
- Vendor-specific messages
- Simple bit-flipping
- Adopted some from Davis work

Execute Fuzzed

1. Poll device
2. Send message

Monitor for Exceptions

1. Check for ack if applicable
2. Poll again
3. If debug, use that
4. If shell, check if service/app still running
5. If TV, will probably notice crash, fun, hard to automate
6. If exception, record msg & state & debug details if avail

DETERMINE EXPLOITABILITY

- This is kind of an adventure unless debug
- Specific to each device

Fuzzing

Complications

- Getting Hold of Devices
  - They are around you however, just need to look
  - Can also emulate w/QEMU + firmware
- Speed
  - 500 bits/s
  - Not much we can do about that
  - Fuzz multiple devices simultaneously
  - RE targets to focus the fuzz

Fuzzing

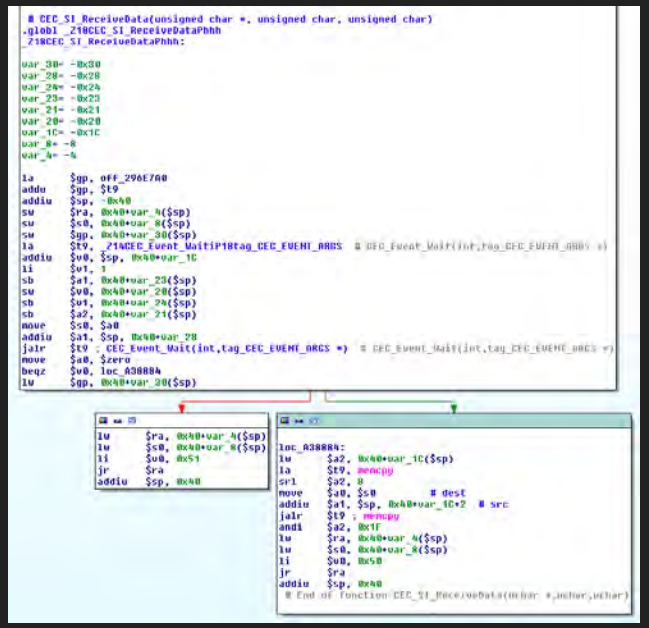
Complications Continued

- Debugging
  - Need to get access to the device
    - Probably no debugger
    - Often painful to compile one for it
  - Collect Data
  - Deduplicate
  - Repro

Targets

Home Theater Devices

- Samsung Blu-ray Player (MIPS)
  - Targeted because already have shell
  - (Thx Ricky Lawshae)
  - Local shell to get on & study device
- Philips Blu-ray Player
- Samsung TV
- Panasonic TV
- Chromecast
- Amazon Fire TV Stick



Targets

Mobile devices

- Kindle Fire
- Galaxy S5 (S6 dropped MHL)
- Galaxy Note
- Chromebook

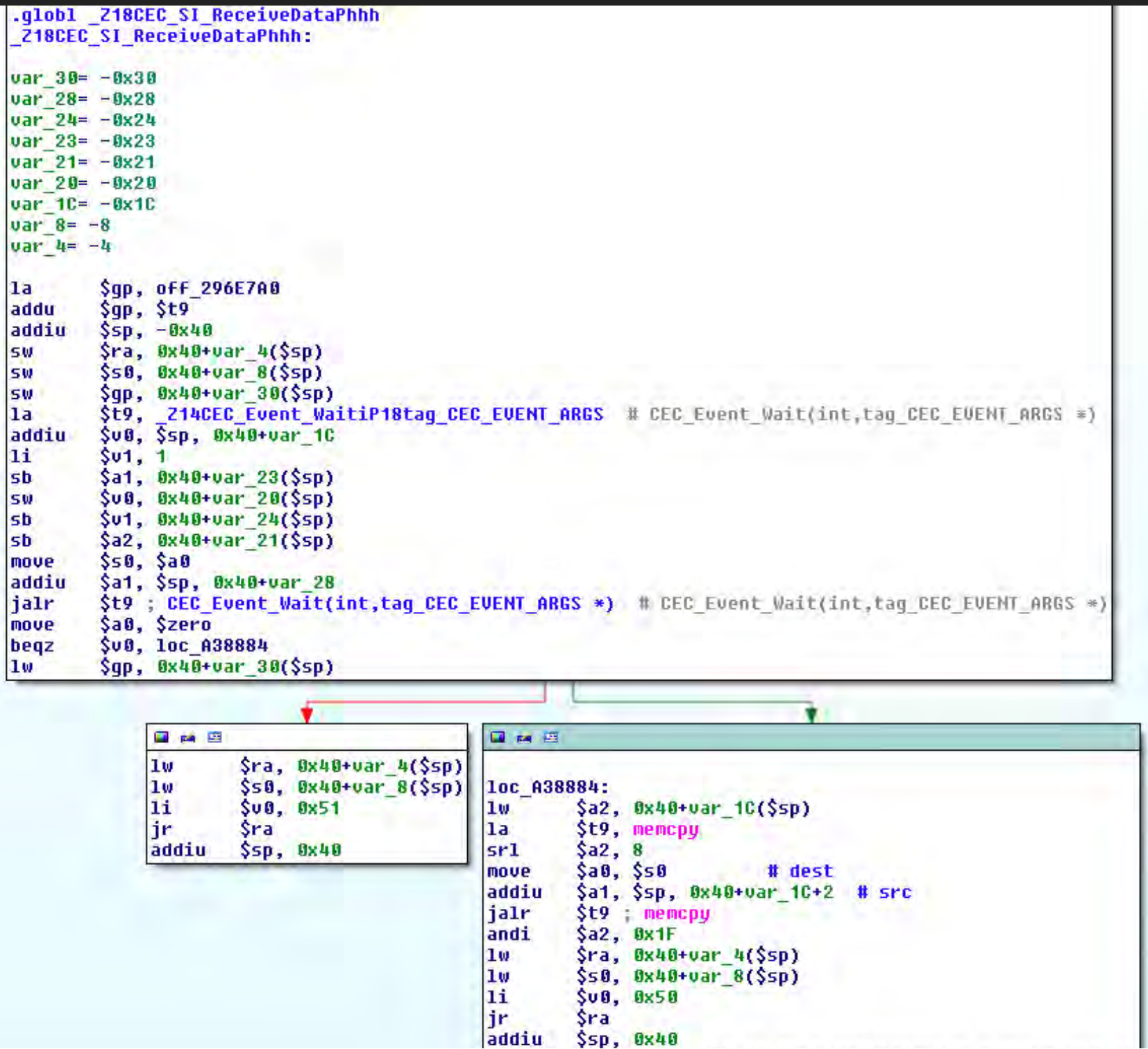
Fuzzing

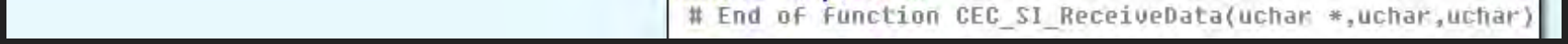
Results

Vulns Discovered

Demos & Videos

- Panasonic TV
- Samsung Blu-ray Player


.globl _Z18CEC_SI_ReceiveDataPhhh
_Z18CEC_SI_ReceiveDataPhhh:
var_30= -0x30
var_28= -0x28
var_24= -0x24
var_23= -0x23
var_21= -0x21
var_20= -0x20
var_1C= -0x1C
var_8= -8
var_4= -4
la $gp, off_296E7A0
addu $gp, $t9
addiu $sp, -0x40
sw $ra, 0x40+var_4($sp)
sw $s0, 0x40+var_8($sp)
sw $gp, 0x40+var_30($sp)
la $t9, _Z14CEC_Event_WaitiP18tag_CEC_EVENT_ARGS # CEC_Event_Wait(int,tag_CEC_EVENT_ARGS *)
addiu $v0, $sp, 0x40+var_1C
li $v1, 1
sb $a1, 0x40+var_23($sp)
sw $v0, 0x40+var_20($sp)
sb $v1, 0x40+var_24($sp)
sb $a2, 0x40+var_21($sp)
move $s0, $a0
addiu $a1, $sp, 0x40+var_28
jalr $t9 ; CEC_Event_Wait(int,tag_CEC_EVENT_ARGS *) # CEC_Event_Wait(int,tag_CEC_EVENT_ARGS *)
move $a0, $zero
beqz $v0, loc_A38884
lw $gp, 0x40+var_30($sp)
lw $ra, 0x40+var_4($sp)
lw $s0, 0x40+var_8($sp)
li $v0, 0x51
jr $ra
addiu $sp, 0x40
loc_A38884:
lw $a2, 0x40+var_1C($sp)
la $t9, memcpy
srl $a2, 8
move $a0, $s0 # dest
addiu $a1, $sp, 0x40+var_1C+2 # src
jalr $t9 ; memcpy
andi $a2, 0x1F
lw $ra, 0x40+var_4($sp)
lw $s0, 0x40+var_8($sp)
li $v0, 0x50
jr $ra
addiu $sp, 0x40



Exploitation

- Background Info
- Barriers
- Samsung TV

Post exploitation

- Enable HEC
- Enable LAN
  - Attack LAN services if nec
  - Enable higher speed exfil etc
- Wake-Over-CEC
- Beachhead for attacking other devices
- Hiding

Future Work

- Explore Attack Surface of
  - HDMI: 3D, Audio Return Channel, more w/HEC
  - Feature adds to CEC
- Moar devices
- Emulation
- Undo bad Python

Conclusion

- Becoming more and more pervasive and invasive
- Old vuln types are new again
- Hard, sometimes impossible, to upgrade, maintain, configure
- Risk = Vulnerabilty x Exposure x Impact
  - The vulns are there
  - Exposure is growing
  - Impact is probably highest for your privacy
- What next? How do we fix or mitigate this?

References

- blackhat.com/bh-eu-12-Davis-HDMI
- github.com/Pulse-Eight/libcec
- hdmi.org
- cec-o-matic.com/
- p8-USB-HDMI-adapter
- Simplified Wrapper & Interface Generator swig.org
- Reveal.js github.com/hakimel/reveal.js