HIGH-DEF FUZZING

EXPLORING VULNERABILITIES IN HDMI-CEC

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

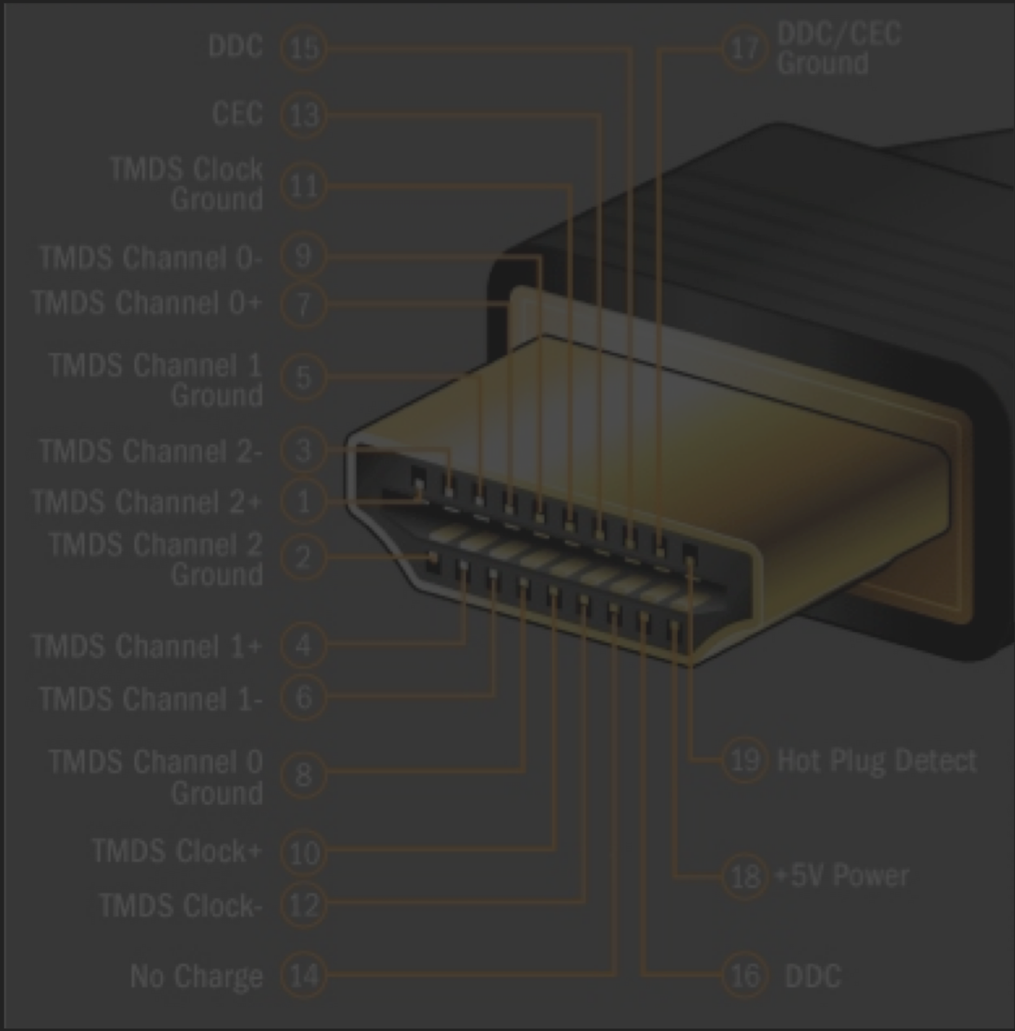
ZERO DAY
INITIATIVE

Which of the following is false?

1. Have had 10 knee surgeries... and 5 others
2. Worked at JHUAPL... did mostly weapon sys assessments
3. Was voted "most athletic" in high school... don't judge a book by its cover ;)
4. Previously ran assessments at the 92d Info. Warfare Aggressor Sq. (USAF)... now 92d Info. Ops. Sq - vuln assessments/pentests/red teams
5. Have a B.S. in Aeronautical Engineering from RPI... Indeed. Also, an MIS & some CS from JHU
6. ~~Am an external Metasploit dev~~... I was, but quit last month
7. Had C2 of 50 nuclear ICBMs on 11 Sep 2001... Interesting story 🍻

Overview

- What is CEC
- Specs & Implementations
- Design Details
- Protocol
- Attack Vectors & Surface
- Fuzzing CEC
- Some Results
- Future Work



Why?

- Wanted to research an area that was relatively untouched
- For me: assembly > C/C++ and RISC > CISC
- Another attack vector for mobile devices via:
  - Mobile High-Definition Link (MHL)
  - Slimport
  - Many car stereos as well
- My son is completely obsessed with cords/wires, esp HDMI

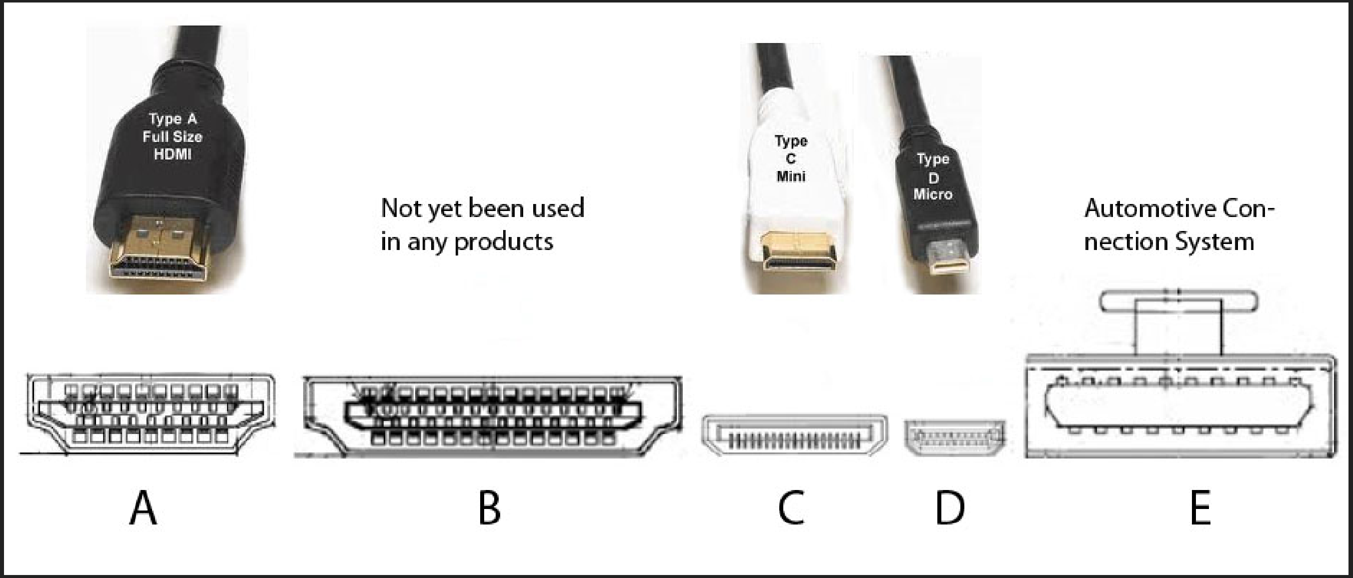
Previous Research

- HDMI – Hacking Displays Made Interesting
  - Andy Davis
  - BlackHat EU 2012
  - GUI Python CEC fuzzer
    - Somewhat simplistic
    - No exception monitoring
    - No crash data gathering

What is HDMI?

High Def Multimedia Interface

- HDMI is an interface specification
- Implemented as cables & connectors
- Successor to DVI



What is CEC?

Consumer Electronics Control

- Feature defined in the HDMI spec
- Allows user to command & control up to 15 devices
- Can relay commands from remotes
- It's what automatically changes your TV input
- Vendor-extendable
- Adopted by some other technologies

That Don't Look Like HDMI!

Still has CEC however

- Slimport
  - Think ~ Amazon, Google, Blackberry, LG G+
- Mobile High-Definition Link (MHL)
  - Think ~ HTC, LG Optimus+, Samsung (not G6)
  - Remote Control Protocol

Specs & Features

History

Ver	Published	Features
1.0	Dec 2002	Boring stuff
1.1	May 2004	Boring stuff
1.2	Aug 2005	Boring stuff
1.2a*	Dec 2005	Fully spec'd CEC

* This is the **good** stuff, for vulnerabilities anyway

Specs & Features

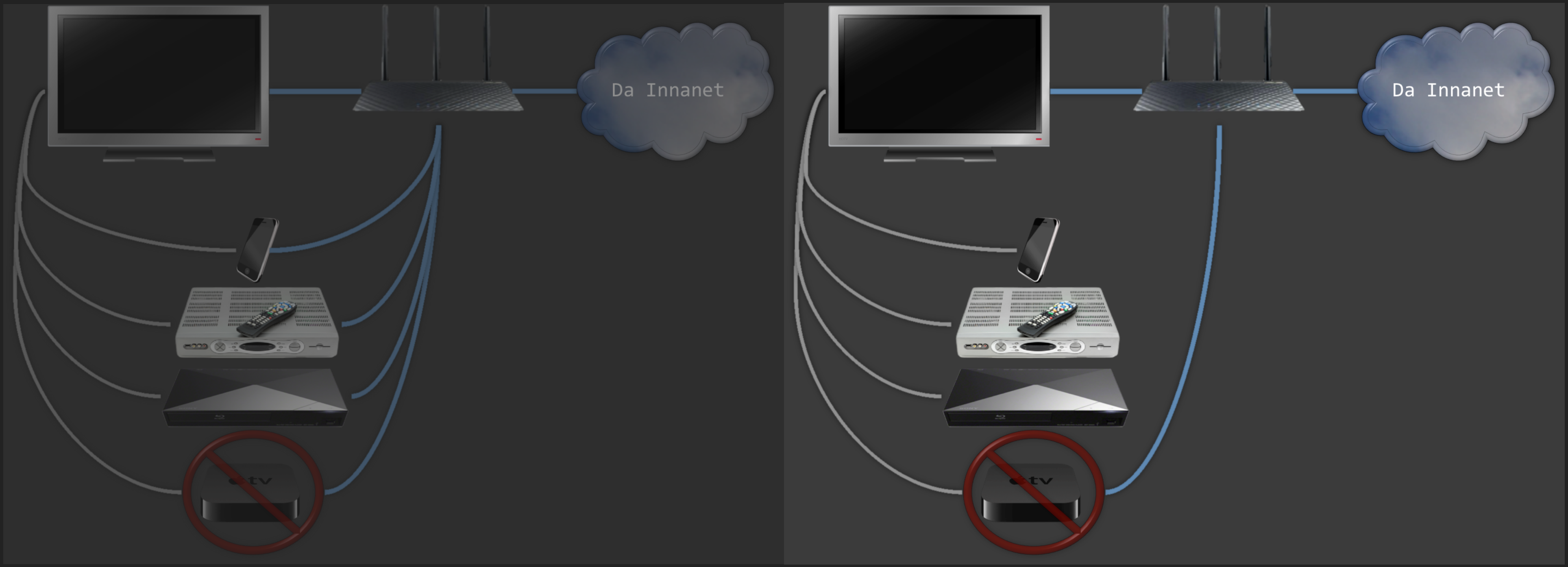
History Continued

Ver	Published	Features
1.3-3c	'06-'08	Whizz-bang A/V & new conns
1.4*	May 2009	Features++: 4k, HEC, ARC, 3D, micro
2.0	Sep 2013	4k @60fps, Dual View, 3D++, CEC++

* **Most widely** deployed & available, more in a sec

Interesting 1.4 Features

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



CEC Details

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

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

CEC Addressing

PHYSICAL

- N.N.N.N where 0x0<=N<=0xF
- Root display (TV) is always 0.0.0.0
- Required as CEC has a notion of switching

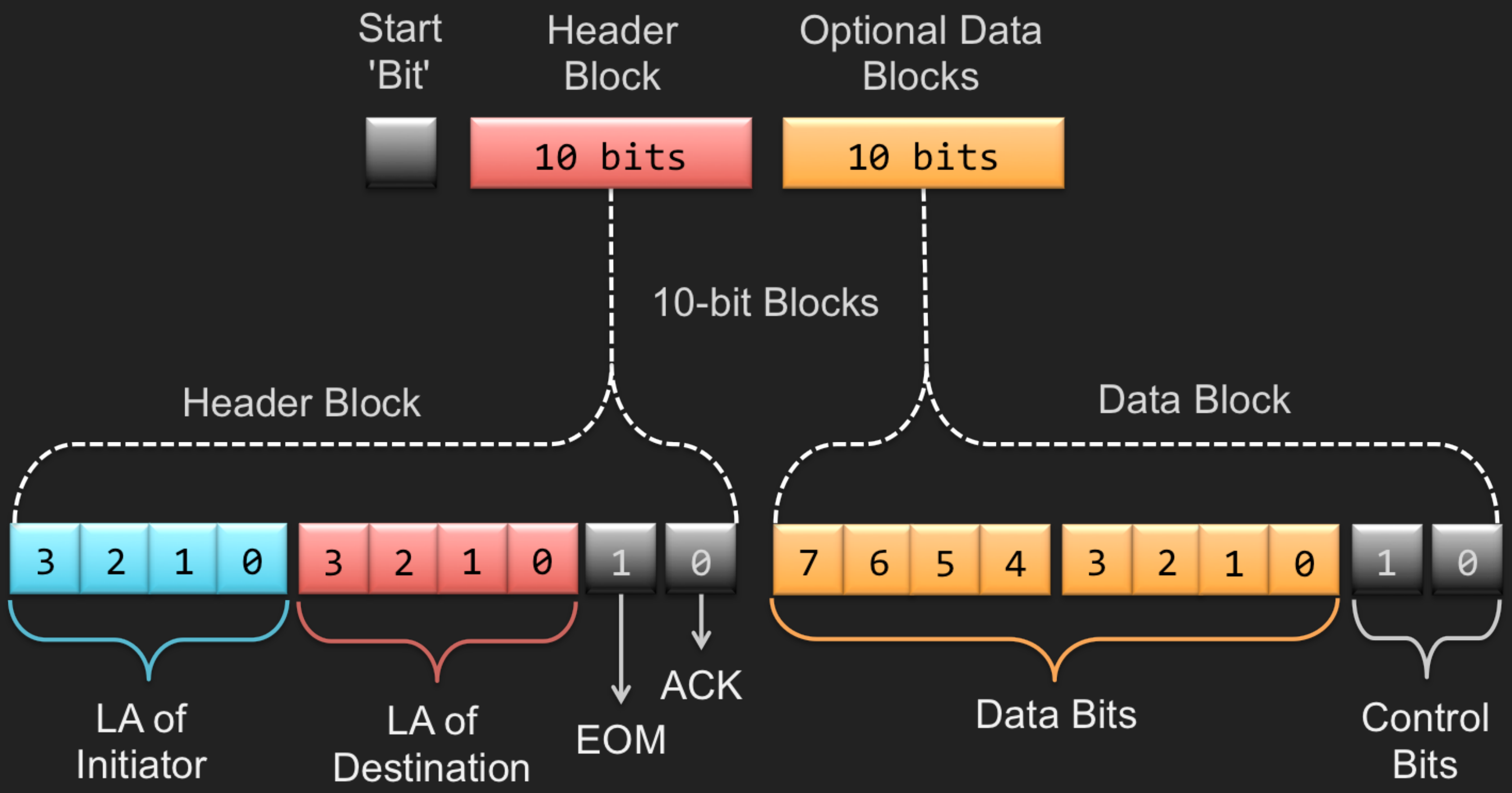
LOGICAL

- L where 0x0<=L<=0xF
- Root display (TV) is always 0
- Negotiated by product type
- Example: first STB in system is always 3

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



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"

Opcode Block

Really just a data block

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

- (8bits) Opcode (Big-endian/MSB first)
- (2bits) Control bits (EoM & Ack)
- Example: 10000010:1:0 = 0x82 (Active Source)

CEC Protocol

The long and short of it...

- 0F - Broadcast ping
- 1 F :82 :10:00

  Source Dest (Bcast) Opcode (Active Src) Param (PA of src)
- 1 0 :64 :40:52:75:78:43:6F:6E:32:30:31:35

  Source Dest (TV) Opcode (Set OSD String) Msg params

  44: Display control flags, rest is ASCII string
- S D :OP :61:41:41:41:41:41:41:41:41:41:41:41:41:41

  Source Dest Opcode Msg params

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

- Big-endian/MSB first
- Text is only printable ASCII (0x20 <= A <= 0x7E)
- Messages can be directly addressed, broadcast, or either
- **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

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
  - Might be useful during fuzzing

Attack Vectors & Thoughts

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

Known popular mobile devices that implement MHL

Attack Surface

- CEC commands
- CEC vendor-specific commands
- HEC commands
- HEC functionality

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 commands?
- 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 Bridge
- 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"))
```

YES. It would appear at least.

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

Fuzzing Process

- It has been done (Davis) 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++

https://media.blackhat.com/bh-eu-12/Davis/bh-eu-12-Davis-HDMI-WP.pdf

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 Data

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

If Shell but !Debugger

- Samsung BluRay Player has BASH
- But not 'watch'
- Fake it:

```
while true; do
  date
  ps aux | grep "[a]pp_player"
  if [ $? -ne 0 ]; then
    # do crash investigation
  fi
  sleep 0.5
done
```

Also TTY Output

```
[API_CECCMD_FeatureAbort] Return value is 0x31
API_CECCMD_FeatureAbort(op:0xB4) start.
[AP_INFOLINK/Fatal] 8:Starting background widget manager !!!
[TCFactory::GetOption] option = 37 value = 0
[TCFactory::GetOption] option = 51 value = 0
[API_CECCMD_FeatureAbort] Return value is 0x36
verified = 1
[AP_INFOLINK/Fatal] 9:CWidgetEngine::createSmartSideBar ret TRUE
[AP_INFOLINK/Fatal] 10:CWidgetEngine::activateSmartSideBar ret TRU
```

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

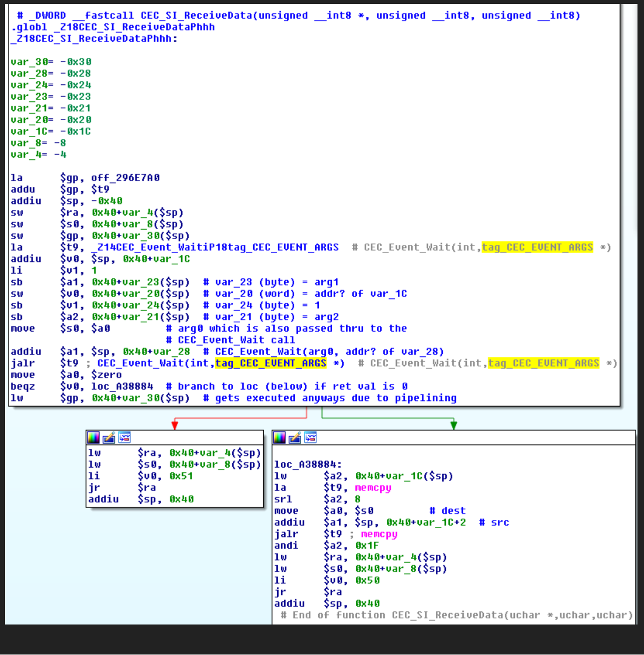
Complications Continued

- Debugging
  - Need to get access to the device
    - Probably no debugger
    - Often painful to compile one for it
    - Keep an eye out for gdbserver files however
  - Collect Data
  - Deduplicate
  - Repro

Targets

Home Theater Devices

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



```
 # _DWORD __fastcall CEC_SI_ReceiveData(unsigned __int8 *, unsigned __int8, unsigned __int8)
.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)  # var_23 (byte) = arg1
sw      $v0, 0x40+var_20($sp)  # var_20 (word) = addr? of var_1C
sb      $v1, 0x40+var_24($sp)  # var_24 (byte) = 1
sb      $a2, 0x40+var_21($sp)  # var_21 (byte) = arg2
move    $s0, $a0            # arg0 which is also passed thru to the
                            # CEC_Event_Wait call
addiu   $a1, $sp, 0x40+var_28  # CEC_Event_Wait(arg0, addr? of 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  # branch to loc (below) if ret val is 0
lw      $gp, 0x40+var_30($sp)  # gets executed anyways due to pipelining
```

```
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
 # End of function CEC_SI_ReceiveData(uchar *,uchar,uchar)
```

Targets

Mobile devices

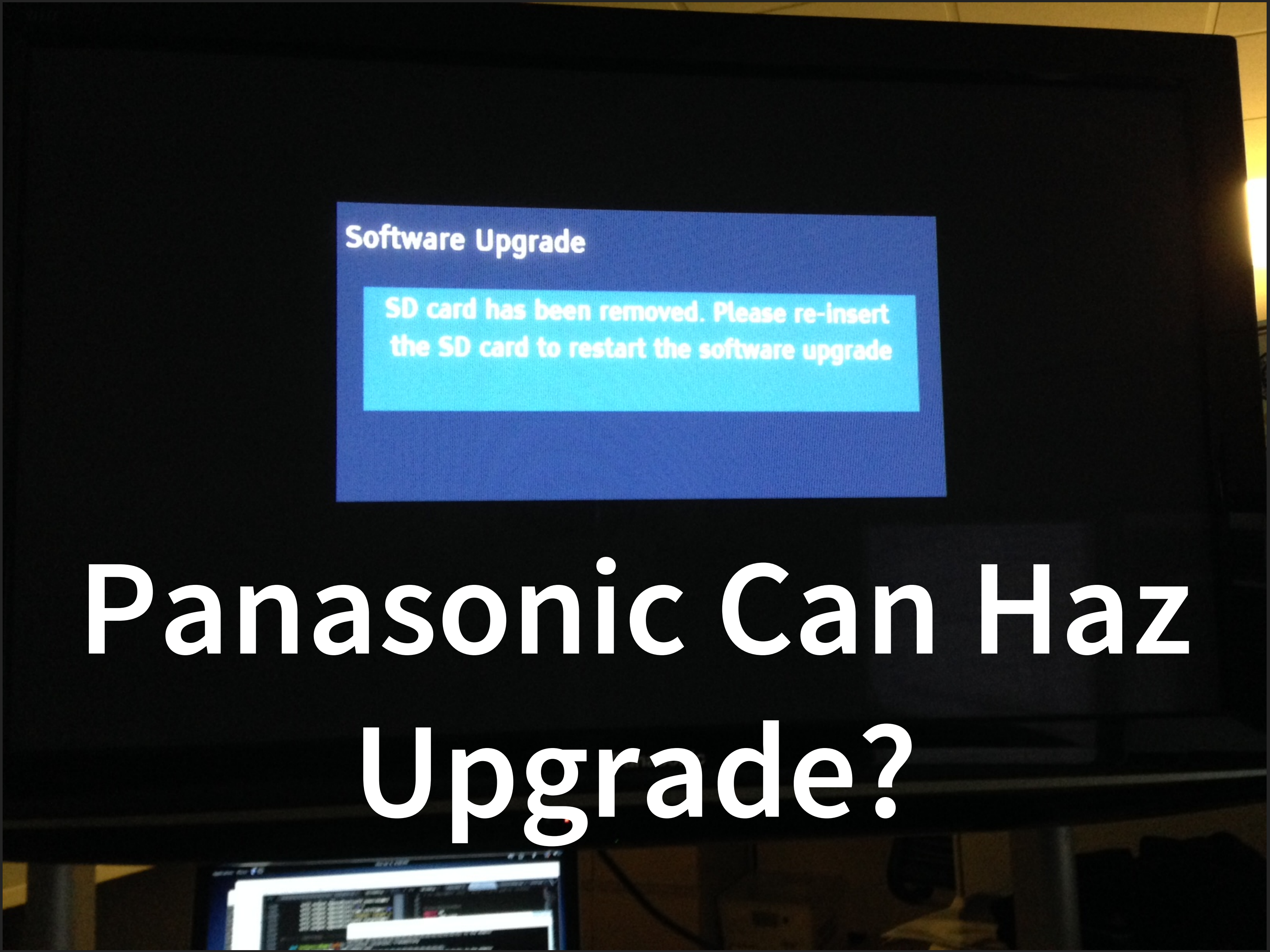
- Kindle Fire
- Galaxy S5 (S6 dropped MHL)
- Galaxy Note
- Chromebook

Results

There's definitely more to be done

Issues Discovered

- Panasonic TV
- Samsung Blu-ray Player



Software Upgrade

SD card has been removed. Please re-insert the SD card to restart the software upgrade

Panasonic Can Haz Upgrade?

Samsung's app_player

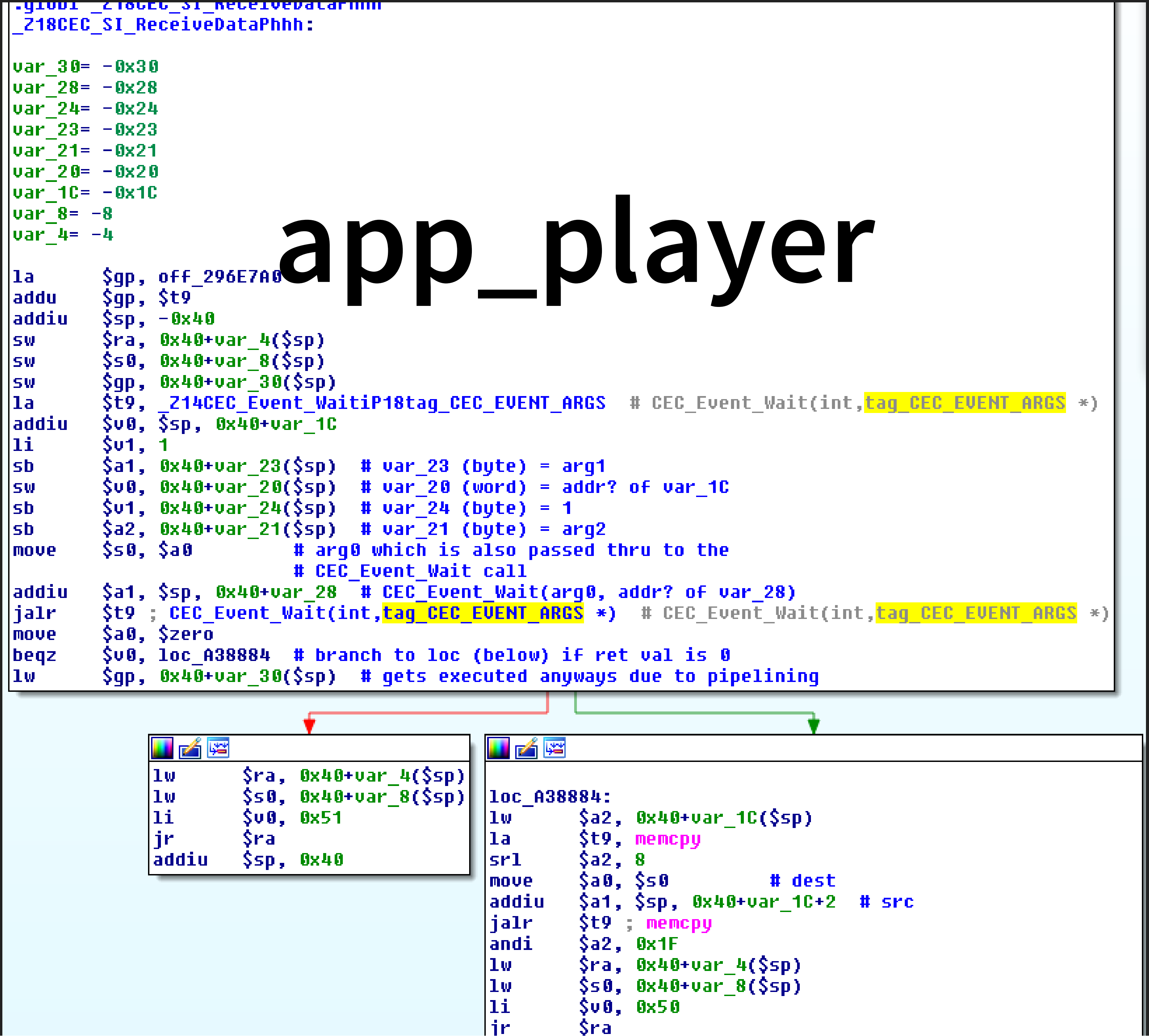
- Handles CEC for BluRay player
- Pulled via Ricky's root shell
- Did some manual RE and
- Rudimentary analysis with some ghetto IDAPython

```
banned = ['memcpy', 'strcpy', 'strncpy', 'etc...']
for func in banned:
  print('Processing ' + func)
  for xref in idautils.CodeRefsTo(idc.LocByName(func), True):
    print(idc.Name(
      idc.GetFunctionAttr(
        xref, idc.FUNCATTR_START
        )) + ' disasm: ' + idc.GetDisasm(xref))
```

Samsung's app_player

- jalr $t9; strcpy => 333
- jalr $t9; strncpy => 409
- jalr $t9; memcpy => 310
- jalr $t9; [.*]printf => 11685
- /me wrings hands
- However, most are not called by CEC code :(
  - 3 memcpy's, 2 of which I had already found manually
  - 73 printf's, but aren't (so far) exploitable conditions

app_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)  # var_23 (byte) = arg1
sw      $v0, 0x40+var_20($sp)  # var_20 (word) = addr? of var_1C
sb      $v1, 0x40+var_24($sp)  # var_24 (byte) = 1
sb      $a2, 0x40+var_21($sp)  # var_21 (byte) = arg2
move    $s0, $a0          # arg0 which is also passed thru to the
                          # CEC_Event_Wait call
addiu   $a1, $sp, 0x40+var_28  # CEC_Event_Wait(arg0, addr? of 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  # branch to loc (below) if ret val is 0
lw      $gp, 0x40+var_30($sp)  # gets executed anyways due to pipelining
```

```
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
```

Post exploitation

- Enable HEC
- Enable LAN
  - Attack LAN services if nec
  - Enable higher speed exfil etc
- Control an MHL device
- Beachhead for attacking other devices
- Hiding

Future Work

- Unuglify my Python
- Integrate into bigger/better fuzz framework
- Exploit CEC & bind shell to network interface
- Exploit CEC, enable HEC, bind shell to HEC interface
- Exploit CEC & "bind" shell to HDMI interface
- Explore attack surface of:
  - HDMI: 3D, Audio Return Channel, more w/HEC
  - Feature adds to CEC (HDMI 2.0)
- Moar devices
- Emulation

Conclusion

- Becoming more and more pervasive and invasive
- Old vuln types may be new again
- May be benefitting simply because code is newer
- Hard, sometimes impossible, to upgrade, maintain, configure
- Risk = Vulnerabilty x Exposure x Impact
  - Exposure is growing
  - Impact is probably highest for your privacy

Links

- github.com/ZDI/hdfuzzing not yet tho
- blackhat.com/bh-eu-12-Davis-HDMI
- github.com/Pulse-Eight/libcec
- hdmi.org
- P8 USB-HDMI Adapter www.pulse-eight.com
- Simplified Wrapper & Interface Generator swig.org
- Reveal.js github.com/hakimel/reveal.js
- cec-o-matic.com

Questions?

ZERO DAY
INITIATIVE

@kernelsmith @thezdi