Implementation of MITM Attack on HDCP-Secured Links

bunnie / 28c3

Twitter your comments @bunniestudios or #28c3
What is HDCP?

- High Definition Content Protection
  - Pixel-level encryption operating at the link layer
- Cipher structure
  - Stream cipher capable of generating 24 bits of pseudorandom data per clock cycle
    - Two parallel 84-bit block functions per round
    - LFSR-based “key scheduler” that whitens block functions at the beginning of each horizontal line of pixels
    - Block functions initialized with publicly exchanged 64-bit initial vector \((A_n)\) that evolves once during each vertical blanking interval
What is HDCP?

- **Key management**
  - Distributed private keys with sort of key revocation
  - Public key is a “key selection vector” (KSV)
    - 40 bits (20 zeros and 20 ones)
  - Private key is a vector of 40 56-bit numbers
  - All private keys derived from a master key consisting of a 40x40 matrix of 56-bit numbers

- **Master key can be directly computed from a collection of 40 unique private keys**
  - The master key was revealed in September 2010
Why HDCP?

- Encrypt video transmissions
  - Complements AACS, BD+ to create studio-to-screen cryptographic chain
- Chain was broken long ago: AACS was the weakest link
  - HDCP master key leak is thus largely a “nop” from the content access standpoint
  - Strippers based on legitimate HDCP keys have long been available on the market; key revocation is largely ineffective
So Why Implement HDCP MITM?

• It’s about control
  – Broadcasters and studios control your screen
  – DMCA and other legal tricks make it illegal for you to modify content – on *your own screen*
So Why Implement HDCP MITM?

- HDCP restricts the implementation of legitimate content manipulation
  - Picture in picture
  - Content overlays
  - 3rd party filtering & image modification

- As a result, there are few HDMI video mixing solutions that can operate on broadcast/movie content
Goal

- Consumer-side content remixing
  - Add web content to existing TV
  - Live comment & chat
- “Over the top” advertising
  - Eliminate ads
  - Or replace ads with targeted ads
- Interactive TV
  - Add interactive elements to broadcast TV
- Compatibility with any TV
How Do We Do It?

bunnie
A’: Intercept and override EDID

- HDMI uses an I2C bus (referred to as DDC) to communicate between video source & sink
- Bus shared between two functions:
  - Monitor capability identification
  - HDCP key exchange
Snoop & squash

- Snooping: intercept key exchange
- Squashing: force TV characteristics
  - The implementation can’t do all HDMI standards
  - Rewrite the EDID record on the fly to reflect only the standards NeTV supports, e.g. no 3D, etc.
I2C snoop & override

MITM hardware

December 29, 2011  28c3  bunnie
I2C snoop & override

I2C clock

oversampling clock

measured clock

I2C data

Override window

Sampling window
I2C snoop & override

- **Oversampled squash** can modify data on the fly
  - Snoop address, and change only bits that need changing
Hot Plug Override

- Hot plug bus has a FET on it to simulate a plug/unplug event
  - Hot plug is an open-drain bus, so this is a safe and easy thing to do
  - Used to resynchronize state when necessary
  - Used to manipulate EDID state
B', C', D': Intercept keys & sync cipher

- Getting $A_n$, AKSV, BKSV accomplished with I2C snooper listening for specific addresses
- Once key exchange is captured, private key vector and shared secret must be derived
  - Final byte write of AKSV is “trigger” to start authentication
  - FPGA fires interrupt to host linux system
  - udev event starts a helper program that does the math
Computing Private Keys

- Modular inner product of master key and public key vectors
  - HDCP master key $K$ is 40x40 matrix of 56-bit numbers
  - $AKSV, BKSV$ are 40-bit numbers consisting of 20 ones and 20 zeros
  - $APK, BPK$ are 40-element vectors of 56-bit numbers
Computing Shared Secret

- **Multiply KSVs by private keys to get 56-bit shared secret $K_m$**

\[ \text{BKSV} \cdot \text{APK} \quad \begin{bmatrix} 0 \\ 1 \\ \cdot \cdot \cdot \\ 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} \text{APK}_{00} \\ \text{APK}_{01} \\ \cdot \cdot \cdot \\ \text{APK}_{38} \\ \text{APK}_{39} \end{bmatrix}^T \quad \begin{bmatrix} 1 \\ 0 \\ \cdot \cdot \cdot \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \text{BPK}_{00} \\ \text{BPK}_{01} \\ \cdot \cdot \cdot \\ \text{BPK}_{38} \\ \text{BPK}_{39} \end{bmatrix}^T \quad = \quad K_m \]
Synchronize Ciphers

- Plug An, Km into cipher hardware
- Init key schedules
- Evolve cipher state based on:
  - Pixclock
  - HSYNC
  - VSYNC
  - Data guardband timings
  - All in plaintext
Pixel-by-pixel synchronization

NeTV

NeTV UI video
Tx-synchronized cipher stream

Swap encrypted pixels for alternate encrypted pixels

Video source
Encrypted video

Video cable

TV

Decrypted video

XOR

Video cable
Synchronize Frame Buffers

- Overlay pixels must be exactly timed to video pixels
- Overlay comes from /dev/fb0 of attached linux computer
- Challenges
  - linux interrupt jitter is too high (10’s to 100’s of us, i.e. thousands of pixels)
  - Local crystal oscillators drift over time (100’s of pixels per frame)
  - Ultimately, overlay “jitters” and “drifts” without tight synchronization
Synchronize Frame Buffers

- Technique #1: source graphics engine pixclock from video, not locally
Synchronize Frame Buffers

- Technique #1: source graphics engine pixclock from video, not locally
- Technique #2: derive timing dynamically from video stream and set /dev/fb0 properties to match
Synchronize Frame Buffers

- Technique #1: source graphics engine pixclock from video, not locally
- Technique #2: derive timing dynamically from video stream and set /dev/fb0 properties to match
- Technique #3: start LCD DMA based on VSYNC start from video stream
Synchronize Frame Buffers

- Technique #1: source graphics engine pixclock from video, not locally
- Technique #2: derive timing dynamically from video stream and set /dev/fb0 properties to match
- Technique #3: start LCD DMA based on VSYNC start from video stream
- Technique #4: add a few video lines’ elastic FIFO buffering to absorb VSYNC interrupt jitter
Chroma Key

- Chroma key reserves a specific color and substitutes its value for “transparent”
- In this implementation, F0,00,F0 (a shade of pink) is the magic color
  - A comparator within the FPGA inspects every pixel and switches a mux
Optimizations

• Key caching
  – Every video source/sink pair has a constant shared secret
  – $K_m$ is cached after first computation to improve system robustness

• EDID caching
  – More important because without EDID caching, users will see a double-blink of the screen
    • First blink is to measure the TV’s capabilities
    • Then we compute the intersection of the TV capabilities and NeTV capabilities
    • Second blink is to override the capabilities we don’t support
A Complete Open Stack

- Provisioning & update server (EC2)
- HTML/javascript widgets (github)
- Webkit
- Angstrom distro (apps/tools)
- U-boot
- Linux
- FPGA (verilog)
- PCB
- Plastics
Application Environment

- TV overlay apps are web pages
  - CSS configured to put “magic pink” as background
  - Apps are javascript/HTML programs
  - But you can extend to any infrastructure that can write to /dev/fb0 (SDL, Flash, etc.)

- Our demo apps are open source and stored in a github repo
  - Updating apps consists of doing a git pull on the client
  - Configured to pull every reboot

- Firmware updates served from EC2 infrastructure
  - Public AMI provided so you can make your own
  - More on this later
HTTP API

- Zeroconf solution for networked interaction with TV
  - API provides method to send events to NeTV
    - So, a smartphone can:
      - Discover NeTV with Bonjour
      - Send events (such as SMS) to the NeTV using HTTP GET
      - NeTV renders these events on your TV
    - Also provides a method for file upload to enable photosharing to the TV
  - Fast, easy integration into “smarthome” environment
  - Example call:
    http://10.0.88.1/bridge?cmd=tickerevent&message=Hello%World&title=Hello%20World
  - Each API call can be restricted to just localhost for security
Turnkey Build System

• Public Amazon EC2 instance with pre-built Angstrom distribution
  – Saves hours of effort downloading & building sources
  – Instance comes configured with local git repo and buildbot to manage builds
  – Built images configured to fetch updates from your own instance
Launching an EC2 AMI

Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its Select button.
## Local cgit repo

### chumby git repositories

Web-based view of EC2 git mirror

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Owner</th>
<th>Idle</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>chumby-oe</td>
<td>shell project</td>
<td></td>
<td></td>
<td>summary log tree</td>
</tr>
<tr>
<td>linux-2.6.28-silvermoon</td>
<td>kernel source</td>
<td></td>
<td>3 weeks</td>
<td>summary log tree</td>
</tr>
<tr>
<td>meta-chumby</td>
<td>chumby openembedded overlay</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>openembedded</td>
<td>openembedded mirror/fork</td>
<td></td>
<td>4 min.</td>
<td>summary log tree</td>
</tr>
<tr>
<td>u-boot-2009.07-silvermoon</td>
<td>u-boot bootloader source</td>
<td></td>
<td>3 weeks</td>
<td>summary log tree</td>
</tr>
</tbody>
</table>

Generated by cgit v0.3.2 at 2011-08-11 09:12:01 (GMT)
### Auto-build triggers based on commits

![Waterfall build status](http://ec2-122-248-194-58.ap-southeast-1.compute.amazonaws.com/waterfall)

#### Waterfall

<table>
<thead>
<tr>
<th>UTC</th>
<th>Changes</th>
<th>Project</th>
<th>State</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:27:31</td>
<td></td>
<td>ntv</td>
<td>building</td>
<td>stdio</td>
</tr>
<tr>
<td>11:27:38</td>
<td></td>
<td>ntv-trunk</td>
<td>build successful</td>
<td>update stdio</td>
</tr>
<tr>
<td>11:26:35</td>
<td></td>
<td>ntv-trunk</td>
<td>idle</td>
<td>update stdio</td>
</tr>
<tr>
<td>Fri 09 Sep 2011 00:37:35</td>
<td></td>
<td></td>
<td>Build 13</td>
<td></td>
</tr>
</tbody>
</table>
Distribute Finished Builds

- Image once, auto-update forever

Index of /output/images/chumby-silvermoon-netv/

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Modified</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Directory/</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>Directory</td>
</tr>
<tr>
<td>Angstrom-chumby-image-eglibc-3.4-v20110703-chumby-silvermoon-netv-testlab/</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>Directory</td>
</tr>
<tr>
<td>Angstrom-chumby-image-eglibc-3.4-v20110703-chumby-silvermoon-netv.Rootfs.EXT3</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>Directory</td>
</tr>
<tr>
<td>boot-chumby-silvermoon-netv.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>config blobs.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>hdmi_720p.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>img-preparing.raw.gz</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>logo.png.gz</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>modules-2.6.28.local-134-chumby-silvermoon-netv نفسها.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>rom-chumby-silvermoon-netv-chumby-image.img.gz</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>u-boot-chumby-silvermoon-netv-local-x11.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>u-boot-chumby-silvermoon-netv.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>zImage-2.6.28-local-x11-chumby-silvermoon-netv.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>zImage-chumby-silvermoon-netv.bin</td>
<td>2011-Sep-08</td>
<td>-</td>
<td>application/octet-stream</td>
</tr>
</tbody>
</table>

lighttpd/1.4.29
Hardware is Open
Plastics are Open
PrOn
And PCB Layout
A Complete Open Solution to HDCP MITM

- **Hardware**
  - Schematics, PCB, industrial design, FPGA

- **Software**
  - Complete, turn-key cloud-based build environment
    - Half an hour from start to production-grade deployment

- **Available at adafruit.com**
  (http://www.adafruit.com/products/609)
Recap: HDCP MITM Implementation

• Complete HDCP MITM solution demonstrated
  – Intercept key exchange on the fly
  – Derive shared secrets & synchronize Tx ciphers
  – Multiplex overlay video using chroma key
  – Avoids decrypting data, therefore DMCA-safe
  – Modifies EDID records to force compatibility

• Enables video compositing functionality
  – Enables unconnected legacy TVs to now have connected TV capabilities
  – Enables you to modify your video content (stop/modify ads, show live internet commentary, etc.)

• A completely open hardware/software stack
Non-Infringing Use of HDCP Master Key!

- Embodiment of a bona-fide, non-infringing and commercially useful application of the HDCP master key
- Blurs the association of the HDCP master key with piracy
  - Prior to this exploit, the only application of the HDCP master key was to circumvent the encryption on copyrighted data
  - Now, there is a non-circumventing application for the HDCP master key