Web-based Attacks on Local IoT Devices

Gunes Acar     Danny Huang     Frank Li
Arvind Narayanan    Nick Feamster
Samsung and Roku Smart TVs Vulnerable to Hacking, Consumer Reports Finds

Security and privacy testing of several brands also reveals broad-based data collection. How to limit your exposure.

Consumer Reports has found that millions of smart TVs can be controlled by hackers exploiting easy-to-find security flaws.

The problems affect Samsung televisions, along with models made by TCL and other brands that use the Roku TV smart-TV platform, as well as streaming devices such as the Roku Ultra.

Call to ban sale of IoT toys with proven security flaws

Natasha Lomas @riptari / Nov 15, 2017
How to reach local IoT devices?

Public devices (e.g., port forwarding)

Local malware

Web attacks (*this paper*)
How to reach local IoT devices?

Public devices (e.g., port forwarding)

Local malware

Web attacks (this paper)
How to reach local IoT devices?

Public devices (e.g., port forwarding)

Local malware

Web attacks (this paper)

1. Discover certain IoT devices
2. Access & control IoT devices
Preparing the Attacks
Targeting HTTP Servers

1. Set up a Raspberry Pi as a WiFi AP, connecting 15 IoT devices and an Android phone.
1. Set up a Raspberry Pi as a WiFi AP, connecting 15 IoT devices and an Android phone.

2. Interact with devices, taking pcaps at the RPi. Observed HTTP endpoints on 7 devices.

<table>
<thead>
<tr>
<th>IoT Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amcrest IP Camera</td>
</tr>
<tr>
<td>D-Link WiFi Camera</td>
</tr>
<tr>
<td>Google Home</td>
</tr>
<tr>
<td>Google Chromecast</td>
</tr>
<tr>
<td>Samsung SmartCam</td>
</tr>
<tr>
<td>Samsung Smart TV</td>
</tr>
<tr>
<td>Belkin Wemo Switch</td>
</tr>
</tbody>
</table>
Targeting HTTP Servers

1. Set up a Raspberry Pi as a WiFi AP, connecting 15 IoT devices and an Android phone.

2. Interact with devices, taking pcaps at the RPi. Observed HTTP endpoints on 7 devices.

3. Searched for further documentation on HTTP APIs
   a. Total: 35 GET, 8 POST

<table>
<thead>
<tr>
<th>IoT Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amcrest IP Camera</td>
</tr>
<tr>
<td>D-Link WiFi Camera</td>
</tr>
<tr>
<td>Google Home</td>
</tr>
<tr>
<td>Google Chromecast</td>
</tr>
<tr>
<td>Samsung SmartCam</td>
</tr>
<tr>
<td>Samsung Smart TV</td>
</tr>
<tr>
<td>Belkin Wemo Switch</td>
</tr>
</tbody>
</table>
Attack 1:
Identify Local IoT Devices
Attack Steps
1. Get local IP (via WebRTC SDP)

192.168.6.6
2. Find active local devices.
   a. Scan local subnet on port 81, sending GET request (via Fetch API)
   b. Measure response times (TCP RST vs TCP timeout)
2. Find active local devices.
   a. Scan local subnet on port 81, sending GET request (via Fetch API)
   b. Measure response times (TCP RST vs TCP timeout)
2. Find active local devices.
   a. Scan local subnet on port 81, sending GET request (via Fetch API)
   b. Measure response times (TCP RST vs TCP timeout)

192.168.6.6

192.168.6.88

TCP SYN to port 81

192.168.6.89
3. Identify IoT devices.
   a. Send request for our GET endpoints to active IP addresses, using HTML5 <audio> element.
   b. Use resulting MediaError message to infer resource availability (*new side channel*).

192.168.6.6

192.168.6.88
Attack Steps

3. Identify IoT devices.
   a. Send request for our GET endpoints to active IP addresses, using HTML5 `<audio>` element.
   b. Use resulting MediaError message to infer resource availability (*new* side channel).

```
192.168.6.6
GET /setup.xml
```

```
192.168.6.88
```

![IoT device and browser symbol]
3. Identify IoT devices.
   a. Send request for our GET endpoints to active IP addresses, using HTML5 <audio> element.
   b. Use resulting MediaError message to infer resource availability (*new* side channel).

**If Exists:** MEDIA_ERR_SRC_NOT_SUPPORTED “DEMUXER_ERROR_COULD_NOT_OPEN: FFmpegDemuxer: open context failed”

**Else:** MEDIA_ELEMENT_ERROR “Format error”
3. Identify IoT devices.
   a. Send request for our GET endpoints to active IP addresses, using HTML5 `<audio>` element.
   b. Use resulting MediaError message to infer resource availability (new side channel).

If Exists: MEDIA_ERR_SRC_NOT_SUPPORTED “Failed to init decoder”
Else: MEDIA_ELEMENT_ERROR “Message 404: Not Found”
3. Identify IoT devices.
   a. Send request for our GET endpoints to active IP addresses, using HTML5 <audio> element.
   b. Use resulting MediaError message to infer resource availability (new side channel).

Safari: Fetches timed out
Edge: No MediaError error messages (Attack 1 does not work)
Implications

Side-channel sidestepping SOP (Chrome bug bounty)

Attack stepping stone

Privacy leaks (e.g., network fingerprinting)
Attack 2:
Access & Control Local Devices
DNS Rebinding

Attack fully bypassing SOP

Requires a web attacker (controls malicious domain + webserver) also controlling domain’s authoritative DNS nameserver
Attack Steps

192.168.6.88
Attack Steps

1. Victim visits *attacker.com*, queries malicious nameserver for *attacker.com*. Return web server IP w/ short TTL.
2. Attacker website loads another resource *test*. 
3. If `attacker.com`’s DNS record is cached, `test` is directly retrieved. If so, wait and retry...
4. If `attacker.com`'s DNS record is *not* cached, browser queries malicious nameserver again. Now return target IP w/ large TTL.
5. This time, retrieving *test* fails. But *attacker.com* is now rebound to the target IP, and can make direct requests.

```
GET /test HTTP/1.1
HTTP 404
```
Attack Steps

5. This time, retrieving test fails. But attacker.com is now rebound to the target IP, and can make direct requests.
Attack on Devices
Attack on Devices

Google Home/Chromecast

Potential attacks:
● Play arbitrary Youtube videos on Chromecast
● Reboot Chromecast/Home
● Scan for WiFi networks and return information
Attack Demo

Attack 3: Detect user’s precise location with Google Home
Implications

Attacker control of IoT device actions

Exploiting IoT device vulnerabilities for full compromise

Privacy leaks (e.g., extensive device fingerprinting or user profiling)
Moving Forward...

- Low barrier to attacks on local IoT devices via malicious websites.
- Need defenses that protect against lateral attacks.
Thank you

https://iot-inspector.princeton.edu/

frankli@cs.berkeley.edu
@frankli714
Attack 1 Countermeasures

Home Users:
- Disable getting local IP via WebRTC SDP
- Configure DHCP to allocate for a larger subnet (e.g., /16)

Browsers:
- Limit private IP access for web pages with public domains

IoT Vendors:
- Respond to all GET request with 200 OK code
Attack on Devices
Attack on Devices

Google Home/Chromecast
Google Home/Chromecast

Access:
- Unique device ID
- Build/firmware version
- SSID of connected WiFi network
- Device schedules/alarms (Home)
Attack on Devices

Google Home/Chromecast

Control:
- Reboot device
- Play any video (Chromecast)
- Scan for WiFi networks and return SSIDs detected
Attack 2 Countermeasures

Home Users:
- Enable DNS forwarding with rebind protection

Browsers:
- Unclear?

IoT Vendors:
- Filter/validate based on HTTP headers

DNS providers:
- Filter private IPs from DNS responses
HTTP endpoints - examples

- **DlinkCamera - GET** http://IP-ADDRESS:80/common/info.cgi
- **Response:**

  - model=DCS-5020L
  - brand=D-Link
  - version=1.14
  - build=9
  - hw_version=A
  - name=DCS-5020L
  - location=
  - macaddr=B0:C5:54:0C:D2:74
  - ipaddr=172.24.1.99
  - netmask=255.255.255.0
  - gateway=172.24.1.1
  - wireless=yes
  - ptz=P,T
  - inputs=0
  - outputs=0
  - speaker=no
  - videoout=no
HTTP endpoints - examples

Get all WiFi networks on WeMo switch:
http://IP-ADDRESS:49154/upnp/control/WiFiSetup1 {
  "method": "POST",
  "body": "<?xml version='1.0'?><SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope'/
<m:GetNetworkList xmlns:m='urn:Belkin:service:WiFiSetup:1'>
"headers": {
  "Content-Type": "text/xml",
  "SOAPAction": "\"urn:Belkin:service:WiFiSetup:1#GetNetworkList\""
}}

Returns all nearby Wifi networks
HTTP endpoints - examples

- **Play arbitrary videos on Google Chromecast** - POST
  http://IP-ADDRESS:8008/apps/YouTube {"method": "POST", "body": "v=oHg5SJYRHA0", "headers": {"User-Agent": "blah"}}

- **Reboot Google Home and Chromecast** -
  http://172.24.1.51:8008/setup/reboot {"method": "POST", "body": "{\"params\": \"now\"}", "headers": {"User-Agent": "blah", "Content-Type": "application/json"}}
## Results

<table>
<thead>
<tr>
<th>IoT Device</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amcrest HD Series IP Security Camera</td>
<td>①</td>
</tr>
<tr>
<td>D-Link Wifi Camera</td>
<td>① ②</td>
</tr>
<tr>
<td>Google Home</td>
<td>① ②</td>
</tr>
<tr>
<td>Google Chromecast</td>
<td>① ②</td>
</tr>
<tr>
<td>Samsung SmartCam HD Pro</td>
<td>① ②</td>
</tr>
<tr>
<td>Samsung UHD Smart TV</td>
<td>① ②</td>
</tr>
<tr>
<td>Belkin Wemo Smart Switch</td>
<td>① ②</td>
</tr>
</tbody>
</table>

Table 1: IoT devices with open HTTP servers, and to which attacks (① and/or ②) they are vulnerable.
### Attack 2

<table>
<thead>
<tr>
<th>Capabilities</th>
<th>C</th>
<th>D</th>
<th>H</th>
<th>S</th>
<th>T</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Software Version or Model</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Get Current SSID</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Get Nearby SSIDs</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Get Device Unique Identifier</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Get Owner’s Username</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Change State</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table 3:** What Attack ② could do to IoT devices: Google [C]hromecast, [D]-Link Camera, Google [H]ome, Samsung [S]martCam, Samsung [T]V, and [W]emo Switch.
Attack 2: Which OSes and browsers are vulnerable

<table>
<thead>
<tr>
<th>OS</th>
<th>Request</th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>GET</td>
<td>C D H S T W</td>
<td>C D H S T W</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>C H T W</td>
<td>C H T W</td>
<td>N/A</td>
</tr>
<tr>
<td>macOS</td>
<td>GET</td>
<td>C D H S T W</td>
<td>C D H S T W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>C H T W</td>
<td>C H T W</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>GET</td>
<td>C D H S T W</td>
<td>C D H S T W</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>C H T W</td>
<td>C H T W</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 4: Which operating systems and browsers were vulnerable to Attack 2 against the following devices: Google [C]hromecast, [D]-Link Camera, Google [H]ome, Samsung [S]martCam, Samsung [T]V, and [W]emo Switch. An unformatted letter indicates that the attack was successful on all known HTTP endpoints on a given device; an underline indicates unsuccessful attacks on all of the HTTP endpoints; and italics indicates that some of the endpoints were vulnerable to our attack. We omit listing Microsoft Edge as all attacks failed on it.
Responsible Disclosure

- We reported the vulnerabilities to...
  - Browser vendors: Chromium (Google), Mozilla
  - IoT vendors: Google, Samsung, D-Link, Belkin
- Chromium offered bug bounty of $500
  - Fixed, will be released in v68
- Mozilla bug is still “Unassigned”
- Google Home: known issue
- Belkin promised to release a patch in August
- Ack from Samsung
- No response from D-Link