# Reverse Engineering of Archer C7 V1 to Analyze and Demonstrate Impactful CVE

Nicholas Cottrell
University of Central Florida
Orlando, FL
nick.cottrell@knights.ucf.edu

Anna Graterol
University of Central Florida
Orlando, FL
agraterolstefanelli@knights.ucf.edu

Mana Mostaani
University of Central Florida
Orlando, FL
mana.mostaani@knights.ucf.edu

## ABSTRACT

Every IoT and tech device has firmware. Firmware gets updates when new features come out or more often, when vulnerabilities are discovered to fix them. The way that professionals discover these vulnerabilities are commonly by reverse engineering the firmware of the given device and following its code. When source code is clearly readable, it can be obvious when a section can be taken advantage of. Attackers take advantage of those vulnerabilities to access personal data from other users. Accessing the network allows them to install a series of malware in which they get benefits to steal users' privacy. Is vital to work on reverse engineering of a firmware as it allows to keep data safe. Finding the vulnerabilities before this can be a huge risk for any user. It understands why and how the firmware is vulnerable and how to approach the solution of it. We want to see exactly what code is running that allows us to take advantage of this device then demonstrate it to support our understanding of the device. In this project we work on understanding how the firmware works, which OS is running, how the CVE works and aspects of the router being vulnerable. We did our tests on the TP Link Archer C7v1 firmware.

## CCS CONCEPTS

• **Security and privacy → Reverse Engineering.**.

## KEYWORDS

Reverse Engineering; Exploitation; IoT

**ACM Reference Format:**
Nicholas Cottrell, Anna Graterol, and Mana Mostaani. 2022. Reverse Engineering of Archer C7 V1 to Analyze and Demonstrate Impactful CVE. In *Foundations of Computer Security and Privacy.* ACM, New York, NY, USA, 6 pages.

## 1 INTRODUCTION

The goal our team faced was to find and analyze as many vulnerabilities on the Archer C7 router as possible. This process has been done many times by many different people. In fact, this whole process is a paid industry where people will analyze IoT Devices and craft exploits for a career. Our team followed a smaller standard model than what is commonly practiced. Most industries would crack at as much source code as they while fuzzing all inputs to find errors. We stayed small and decided to extract the firmware code and use individual applications as a base to start looking for vulnerabilities. From there, we would either search for the applications source code or attempt to disassemble the binary ourselves. Once we had our frame of vulnerability, we would work on crafting an exploit. The final product would be our exploit module.

Our team ended of finding two exploits as well as a password hash. The exploits were designated CVE-2011-0762 [3] and CVE-2015-3035 [5] respectively. CVE-2011-0762 would end up being a Denial of Service (DOS) attack on vsftpd while CVE 2015-3035 is a directory traversal attack on Apache httpd.

## 2 RELATED WORK

In this section, we show a comparison of other methods that analyze and demonstrate impactful CVEs in firmware. Some of those related research went through the first initial steps as we did on our own research, such as utilizing binwalk for file extraction. One document described ways of carving a bin file that is utilized by the same router as our firmware [10].

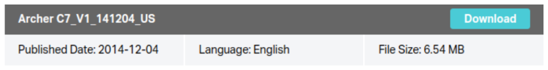
The key difference is that the article utilizes a version of openwrt firmware rather than the default firmware provided by the manufacturer. They also carve out extra data sections, such as the router's kernel, which we never do. Though our path ended up being wildly different to the work done in embedded bits, they provided some good initial insight and direction into how we would get our data.

## 3 APPROACH

Since our approach is similar to the standard model that most Vulnerability Researchers follow, we took our approach in the following practice.

### 3.1 Firmware Extraction

We started at the very first stage which was scouting out our data. We needed to know how the router worked inside and out in order to find bugs within it and from there exploits.


Archer C7_V1_141204_US

Published Date: 2014-12-04 Language: English File Size: 6.54 MB

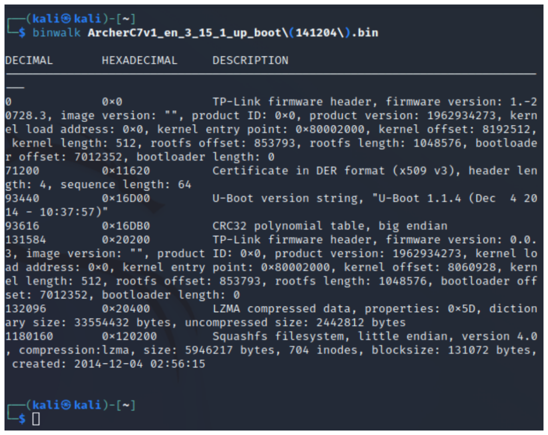
**Figure 1: Downloading the Archer C7_V1_141204_US**

The firmware for the router can be found on TP-Links official website. We decided to go with the earliest firmware version possible. The reason behind this was because we needed as many chances to find bugs as possible. Since this was our teams first attempt at

Vulnerability Research, we need to give ourselves as many opportunities as possible, including for vulnerabilities that already existed. By using the oldest firmware that is offered, we allow for as many unpatched vulnerabilities to be usable.

We needed to get the firmware file from the website [1]. The file given is a zip file that we can unzip to get our bin file. If we wanted to, we could flash the bin file onto our router by going through the update process.

*3.1.1 Binwalk.* However, we want to go further than that. We want to see what is on the filesystem. In order to do that, we need to utilize the tool, binwalk. When running binwalk on the bin file, it gives us information on the files and data that are currently present.



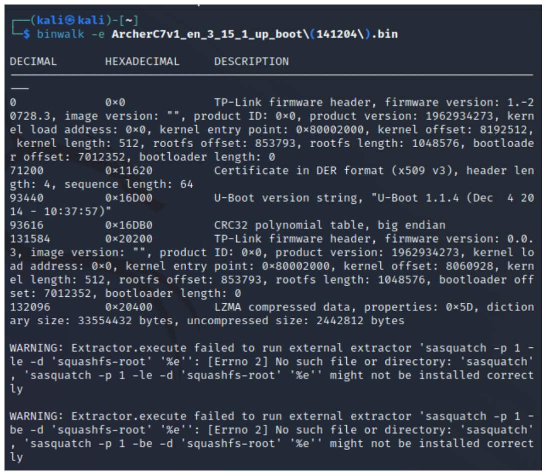
```
┌──(kali㉿kali)-[~]
└─$ binwalk ArcherC7v1_en_3_15_1_up_boot\(141204\).bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
──
0             0×0             TP-Link firmware header, firmware version: 1.-2
0728.3, image version: "", product ID: 0×0, product version: 1962934273, kern
el load address: 0×0, kernel entry point: 0×80002000, kernel offset: 8192512,
 kernel length: 512, rootfs offset: 853793, rootfs length: 1048576, bootloade
r offset: 7012352, bootloader length: 0
71200         0×11620         Certificate in DER format (x509 v3), header len
gth: 4, sequence length: 64
93440         0×16D00         U-Boot version string, "U-Boot 1.1.4 (Dec  4 20
14 - 10:37:57)"
93616         0×16DB0         CRC32 polynomial table, big endian
131584        0×20200         TP-Link firmware header, firmware version: 0.0.
3, image version: "", product ID: 0×0, product version: 1962934273, kernel lo
ad address: 0×0, kernel entry point: 0×80002000, kernel offset: 8060928, kern
el length: 512, rootfs offset: 853793, rootfs length: 1048576, bootloader off
set: 7012352, bootloader length: 0
132096        0×20400         LZMA compressed data, properties: 0×5D, diction
ary size: 33554432 bytes, uncompressed size: 2442812 bytes
1180160       0×120200        Squashfs filesystem, little endian, version 4.0
, compression:lzma, size: 5946217 bytes, 704 inodes, blocksize: 131072 bytes,
 created: 2014-12-04 02:56:15


┌──(kali㉿kali)-[~]
└─$ 
```

**Figure 2: Running binwalk on the bin file**

The two sections that seem interesting to investigate are the TP-Link Firmware Headers at 0x0 and 0x20200, and the squashfs filesystem at 0x120200. The squashfs server seems most interesting since that could contain the linux filesystem that the router runs on. In order to extract the filesystem to something useful, we just need to run binwalk -e on the bin file. We tried to run this, but we got an error on our distribution.



```
┌──(kali㉿kali)-[~]
└─$ binwalk -e ArcherC7v1_en_3_15_1_up_boot\(141204\).bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
──
0             0×0             TP-Link firmware header, firmware version: 1.-2
0728.3, image version: "", product ID: 0×0, product version: 1962934273, kern
el load address: 0×0, kernel entry point: 0×80002000, kernel offset: 8192512,
 kernel length: 512, rootfs offset: 853793, rootfs length: 1048576, bootloade
r offset: 7012352, bootloader length: 0
71200         0×11620         Certificate in DER format (x509 v3), header len
gth: 4, sequence length: 64
93440         0×16D00         U-Boot version string, "U-Boot 1.1.4 (Dec  4 20
14 - 10:37:57)"
93616         0×16DB0         CRC32 polynomial table, big endian
131584        0×20200         TP-Link firmware header, firmware version: 0.0.
3, image version: "", product ID: 0×0, product version: 1962934273, kernel lo
ad address: 0×0, kernel entry point: 0×80002000, kernel offset: 8060928, kern
el length: 512, rootfs offset: 853793, rootfs length: 1048576, bootloader off
set: 7012352, bootloader length: 0
132096        0×20400         LZMA compressed data, properties: 0×5D, diction
ary size: 33554432 bytes, uncompressed size: 2442812 bytes

WARNING: Extractor.execute failed to run external extractor 'sasquatch -p 1 -
le -d 'squashfs-root' '%e'': [Errno 2] No such file or directory: 'sasquatch'
, 'sasquatch -p 1 -le -d 'squashfs-root' '%e'' might not be installed correct
ly

WARNING: Extractor.execute failed to run external extractor 'sasquatch -p 1 -
be -d 'squashfs-root' '%e'': [Errno 2] No such file or directory: 'sasquatch'
, 'sasquatch -p 1 -be -d 'squashfs-root' '%e'' might not be installed correct
ly
```

**Figure 3: The error we faced running the binwalk**

This is because sasquatch does not exist on the system. This was consistent amongst all of the members. After some research, it was apparent that there wasn't a linux package that contained the sasquatch binary, so we had to go to the GitHub and compile and install the sasquatch binaries manually. In order to get the tool working, we had to run the following commands.

```
sudo apt install build-essential liblzma-dev liblzo2-dev zlib1g-dev squashfs-tools
git clone https://github.com/devttys0/sasquatch.git
cd sasquatch; ./build.sh
```

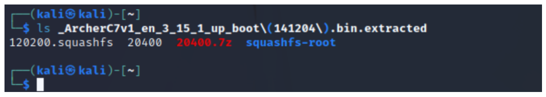
**Figure 4: Fixing the error**

After this, binwalk was able to extract all the data out of our binary without any trouble. We unfortunately still had trouble with installing squashfs because the build script was failing to compile. To fix this, we just installed from a different repo of the same tools.

```
git clone https://github.com/threadexio/sasquatch.git
```

**Figure 5: Installing squashfs**

*3.1.2 Squashfs-root Filesystem.* Afterwards, binwalk will extract the binary and provide us with a squash-root folder. After some investigation, this folder appears to be the system structure of the router. With this, we have a perspective as if we were root within the system, able to see all the configuration files and binaries at our liesure.



```
┌──(kali㉿kali)-[~]
└─$ ls _ArcherC7v1_en_3_15_1_up_boot\(141204\).bin.extracted
120200.squashfs  20400  20400.7z  squashfs-root

┌──(kali㉿kali)-[~]
└─$ 
```

**Figure 6: Squashfs-root folder**

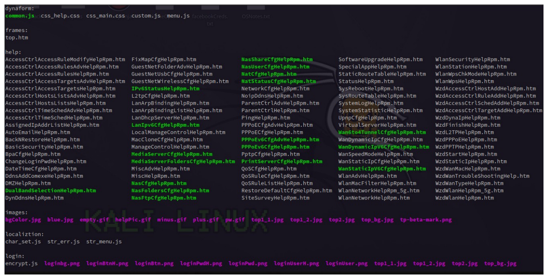
During our search, one of the first places that we checked was /etc/passwd and /etc/shadow. Looking at the directory, passwd was a symbolic link to somewhere. Unfortunately, binwalk removed any symbolic absolute links for safety, so we are unsure where the real passwd file is on the root system. However, the shadow file was not a link, but an actual file. And on further inspection, we discovered that it had content.

This hash for the root password could potentially at minimal give us a means of escalation if we manage to get a shell within the system for further analysis. If we are exceptionally lucky, it could provide us with a means of logging in directly as root onto the system with minimal effort.

Looking in other corners of the filesystem, another advantageous look is the wireless configs found at */etc/ath/wsc_config.txt*. In /etc were also able to find pre initialized keys for dhcpd, which could give use a chance to impersonate the dhcpd server. This would provide little benefit, but it is another potential avenue to look a in terms of bugs and "vulnerabilities."

```
root:$1$GTN.gpri$DlSyKvZKMR9A9Uj9e9wR3/:15502:0:99999:7:::
```

**Figure 7**



```
dynaform:
common.js  css_help.css  css_main.css  custom.js  menu.js

frames:
top.htm

help:
AccessCtrlAccessRuleModifyHelpRpm.htm  FixMapCfgHelpRpm.htm                 NasShareCfgHelpRpm.htm         SoftwareUpgradeHelpRpm.htm        WlanSecurityHelpRpm.htm
AccessCtrlAccessRulesAdvHelpRpm.htm    GuestNetFolderAdvHelpRpm.htm         NasUserCfgHelpRpm.htm          SpecialAppHelpRpm.htm             WlanStationHelpRpm.htm
AccessCtrlAccessRulesHelpRpm.htm       GuestNetUsbCfgHelpRpm.htm            NatCfgHelpRpm.htm              StaticRouteTableHelpRpm.htm       WlanWpsChkModeHelpRpm.htm
AccessCtrlAccessTargetsAdvHelpRpm.htm  GuestNetWirelessCfgHelpRpm.htm       NatStatusCfgHelpRpm.htm        StatusHelpRpm.htm                 WlanWpsHelpRpm.htm
AccessCtrlAccessTargetsHelpRpm.htm     IPv6StatusHelpRpm.htm                NetworkCfgHelpRpm.htm          SysRebootHelpRpm.htm              WzdAccessCtrlHostAddHelpRpm.htm
AccessCtrlHostsListsAdvHelpRpm.htm     L2tpCfgHelpRpm.htm                   NoipDdnsHelpRpm.htm            SysRouteTableHelpRpm.htm          WzdAccessCtrlRuleAddHelpRpm.htm
AccessCtrlHostsListsHelpRpm.htm        LanArpBindingHelpRpm.htm             ParentCtrlAdvHelpRpm.htm       SystemLogHelpRpm.htm              WzdAccessCtrlSchedAddHelpRpm.htm
AccessCtrlTimeSchedAdvHelpRpm.htm      LanArpBindingListHelpRpm.htm         ParentCtrlHelpRpm.htm          SystemStatisticHelpRpm.htm        WzdAccessCtrlTargetAddHelpRpm.htm
AccessCtrlTimeSchedHelpRpm.htm         LanDhcpServerHelpRpm.htm             PingHelpRpm.htm                UpnpCfgHelpRpm.htm                WzdDynaIpHelpRpm.htm
AssignedIpAddrListHelpRpm.htm          LanIpv6CfgHelpRpm.htm                PPPoECfgAdvHelpRpm.htm         VirtualServerHelpRpm.htm          WzdFinishHelpRpm.htm
AutoEmailHelpRpm.htm                   LocalManageControlHelpRpm.htm        PPPoECfgHelpRpm.htm            Wan6to4TunnelCfgHelpRpm.htm       WzdL2TPHelpRpm.htm
BackNRestoreHelpRpm.htm                MacCloneCfgHelpRpm.htm               PPPoEv6CfgAdvHelpRpm.htm       WanDynamicIpCfgHelpRpm.htm        WzdPPPoEHelpRpm.htm
BasicSecurityHelpRpm.htm               ManageControlHelpRpm.htm             PPPoEv6CfgHelpRpm.htm          WanDynamicIpV6CfgHelpRpm.htm      WzdPPTPHelpRpm.htm
BpaCfgHelpRpm.htm                      MediaServerCfgHelpRpm.htm            PptpCfgHelpRpm.htm             WanSpeedModeHelpRpm.htm           WzdStartHelpRpm.htm
ChangeLoginPwdHelpRpm.htm              MediaServerFoldersCfgHelpRpm.htm     PrintServerCfgHelpRpm.htm      WanStaticIpCfgHelpRpm.htm         WzdStaticIpHelpRpm.htm
DateTimeCfgHelpRpm.htm                 MiscAdvHelpRpm.htm                   QoSCfgHelpRpm.htm              WanStaticIpV6CfgHelpRpm.htm       WzdWanMacHelpRpm.htm
DdnsAddComexeHelpRpm.htm               MiscHelpRpm.htm                      QoSRuleCfgHelpRpm.htm          WlanAdvHelpRpm.htm                WzdWanTroubleShootingHelp.htm
DMZHelpRpm.htm                         NasCfgHelpRpm.htm                    QoSRuleListHelpRpm.htm         WlanMacFilterHelpRpm.htm          WzdWanTypeHelpRpm.htm
DualBandSelectionHelpRpm.htm           NasFoldersCfgHelpRpm.htm             RestoreDefaultCfgHelpRpm.htm   WlanNetworkHelpRpm_5g.htm         WzdWlanHelpRpm_5g.htm
DynDdnsHelpRpm.htm                     NasFtpCfgHelpRpm.htm                 SiteSurveyHelpRpm.htm          WlanNetworkHelpRpm.htm            WzdWlanHelpRpm.htm

images:
bgColor.jpg  blue.jpg  empty.gif  helpPic.gif  minus.gif  plus.gif  pw.gif  top1_1.jpg  top1_2.jpg  top2.jpg  top_bg.jpg  tp-beta-mark.png

localiztion:
char_set.js  str_err.js  str_menu.js

login:
encrypt.js  loginbg.png  loginBtnH.png  loginBtn.png  loginPwdH.png  loginPwd.png  loginUserH.png  loginUser.png  top1_1.jpg  top1_2.jpg  top2.jpg  top_bg.jpg
```

**Figure 8: Contents of /web**

When checking the /root folder it turns out to be empty. Same with all the standard system device folders such as /sys, /proc, /mnt, and /tmp. /usr only contained binaries which we will get to soon. The folder that seemed to pop out the most was a */web* folder.

On further inspection, the */web* folder contains all the web source code for the router. The website is HTML, so no potential is a CrossSite Scripting (XSS) Attack or a break in the serverside code since HTML is created on the clients computer. What does seem interesting at first is */web/login/encrypt.js* and */web/oem/model.conf*. On a first glance at encrypt.js, it becomes pretty clear that it does its encryption using md5. This means that web could potentially break the password using a md5 collision attack, or if we were to sniff the actual password, we could bruteforce the hash really easily. When looking at the model.conf file, it cant be opened normally like a text file. Linux calls the file just data, and the data headers do not point to anything obvious. This gave our team two theories. The first one is that it is custom data that is meant to integrate into a program similar to a database file. The second theory is that it is a normal plain text file that has been encrypted and can only be read when the data is decrypted with a private key stored somewhere within the router. Without more information, we cannot pursue either possibility.

## 3.2 Application versioning

After digging at the files that we found within our filesystem. It was time to look at the individual applications within the system. The first thing that our team wanted was the version kept within each binary. Knowing the guaranteed versions of the applications helps make exploit research easier as we're not going off the guesses that our scanners use on the router. It can also be beneficial to potentially run a local exploit if we manage to get some form of shell running in the system.

Since we knew that our binaries were held in /bin, /sbin, /usr/bin, and /usr/sbin, we just had to go down the list and look for binary versions. The simplest and fastest approach with this would be to strings the binaries. Most binaries will include some version of help argument. Because of this, most binaries keep their version number in string format for easy printing. Unfortunately strings prints all the strings within the binary, including strings for other things, or even bytes that could translate to ascii characters. In order to keep our search small, we had to specify it with grep. We then needed a regular expression to select lines that were in a standard version format.

| | |
|---|---|
| bin/busybox | 1.01 |
| bin/cat | 0.9.9-pre |
| bin/chmod | 0.9.9-pre |
| bin/date | 0.9.9-pre |
| bin/df | 0.9.9-pre |
| sbin/apstats | v0.1 |
| sbin/hostapd | 2.0-devel |
| sbin/ip6tables | 1.4.5 |
| sbin/ip6tables-multi | 1.4.5 |
| sbin/ip6tables-restore | 1.4.5 |
| sbin/ip6tables-save | 1.4.5 |
| sbin/iptables | 1.4.5 |
| sbin/iptables-multi | 1.4.5 |
| sbin/iptables-restore | 1.4.5 |
| sbin/iptables-save | 1.4.5 |
| sbin/klogd | 0.9.9-pre |
| sbin/logread | 0.9.9-pre |
| sbin/lsmod | 0.9.9-pre |
| sbin/reboot | 0.9.9-pre |
| sbin/rmmod | 0.9.9-pre |
| sbin/route | 0.9.9-pre |
| sbin/ssdk_sh | 2.6 |
| sbin/syslogd | 0.9.9-pre |
| sbin/tphotplug | 1-1.1;1-1.2 |
| sbin/udhcpc | 0.9.9-pre |
| sbin/vconfig | 0.9.9-pre |
| usr/bin/[ | 0.9.9-pre |
| usr/bin/arping | 0.9.9-pre |
| usr/bin/dropbear | 2012.55 |
| usr/bin/dropbearkey | 2012.55 |
| usr/bin/httpd | 2.6.31;1.1.4 |
| usr/bin/lld2d | 1.2 |
| usr/bin/logger | 0.9.9-pre |
| usr/bin/test | 0.9.9-pre |
| usr/bin/tftp | 0.9.9-pre |
| usr/bin/top | 0.9.9-pre |
| usr/bin/uclited | 2.6.31;3.0.10 |
| usr/bin/vsftpd | 2.3.2 |
| usr/sbin/bpalogin | 2.0.2 |
| usr/sbin/smbd | 3.1a |
| usr/sbin/udhcpd | 0.9.9-pre |
| usr/sbin/xl2tpd | xl2tpd-1.1.12 |

**Table 1: Binaries and their versions**

[language=bash] for binary in /bin/*,/sbin/*,/usr/bin/*,/usr/sbin/*; do echo $binary strings $binary | grep -E [0-9]+([̇0-9]+)+ done

After running that bash snippet, grabbing versions was easy. After all of that, we were able to log all our application versions in Table 1. This table would become helpful when it came to researching for vulnerabilities since a majority of them only work on specific versions.

### 3.2.1 *Reversing engineering with Ghidra.*

Some applications were not as easy to get a version out of them. Some simply kept their version in a single or series of numbers. In order to find these, we had to get access to the source code. Without the version, we could not simply attempt to download the source code off the internet. Because of this, we had to do things the hard way and reverse the binaries.

There are multiple reverse engineering tools that we could have used. Since this was our teams first time doing reverse engineering, we decided to use Ghidra since it is a popular and rather simple reverse engineering tool. Once we put ghidra through the selected binaries, we had to do quite a bit of reading through the pseudo source code generated in order to understand what the code was doing and decipher what a particular variable was being used for.

This was a small part, but this would become useful after our research for vulnerabilities. This is because we would use Ghidra in order to dig further into the binaries and find detailed code potentially describing how it failed.

## 3.3 Vulnerability Research

Looking at the table from Table 1, some binaries that popped out at us were */usr/bin/httpd*, */usr/bin/vsftpd*, and */usr/bin/smbd*. The reason that these popped out at us was because all these versions seemed low and they were all binaries that used internet connection.

The first place that we checked was metasploits library. The only exploit we found of interest was in *exploit/unix/ftp/vsftpd_234_backdoor* [4] The exploit depicts a potential backdoor found in version 2.3.4. It seemed like a shot in the dark for us since it depicted a newer version. We checked it anyway and unfortunately, the application with our firmwares current version was not vulnerable. The other vulerability that we found that seemed interesting was *exploit/linux/misc/tplink_archer_a7_c7_lan_rce* [6]. Unfortunately this also ended up being a dud.

The next place we searched was the exploit database. There we found a specific exploit of ID: 16270 [9]. Further reading into the exploit, it mentions CVE-2011-0862 [3]. After trying this exploit from the source file given, we found that the exploit worked as intended.

We managed to lose how we found this CVE to begin with, but the next CVE we were able to discover was CVE-2015-3035 [5]. Through mitres website, we found another website going into greater detail on the vulnerability including a Proof of Concept (PoC) of the exploit [11]. Nick tested this by bringing up Burpsuite and sending a customized HTTP request for the */etc/passwd* file. The result was the contents of said file.

With at least two confirmed vulnerabilities, the next step was to dig deeper into the applications these vulnerabilities affected and discover what allows them to exist in the first place.

## 3.4 Digging deeper into the binaries

Using data gathered from MITRE on CVE-2011-0762 [3] and Ghidra, we were able to look deeper into the binary at the vsf_filename_passes_filter function. Unfortunately, a lot of the data within the binary was stripped away which would have made guessing a really tough time for us. Luckily, we were able to find an archive of the binary online [2]. With that we had a means of explaining the vulnerability later on.

## 3.5 Exploit Writing

Now that we had the vulnerabilities and we could pinpoint where the vulns were coming from, it was time to write a Proof of Concept for each of them. Because Nick knew ruby pretty well, our team wrote the exploits to run on the Metasploit framework.

*3.5.1 Writing exploits with Metasploit.* After hours of writing and using other exploits as templates, the exploits can be found on Nicks fork of the framework or within the framework depending on if rapid7 decided to merge the modules into the official repo.

## 4 EVALUATION

## 4.1 Binwalk loot

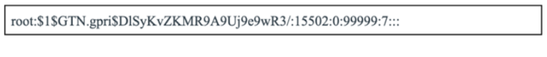
After looking through all the files, we found several points that seemed worth noting. To clarify some findings, the router can be confirmed to be using a linux filesystem. It does not appear to be any flavor or distro of linux, but the router uses a mips architecture. During our search, one of the first places that we checked was /etc/passwd and /etc/shadow. Looking at the directory, passwd was a symbolic link to somewhere. Unfortunately, binwalk removed any symbolic absolute links for safety, so we are unsure where the real passwd file is on the root system. However, the shadow file was not a link, but an actual file. And on further inspection, we discovered that it had content.

After running the hash through john with a wordlist starting with the router's known default password and rockyou, we discovered that the password installed within the firmware was not the router's default password of admin, but rather sohoadmin. This is interesting because it is not the reported default password for the router, so potentially this can be utilized as long as it does not get changed in the middle of an operation.

## 4.2 CVE-2011-0762

Our next step was to look at the binaries that are present on the system. If we could get the versions of an interesting binary, we could find potential exploits for the router. After a good amount of searching, we found what we were looking for. The binary /usr/bin/vsftpd. This one seemed interesting to us because there is a well-known backdoor for vsftpd on version 2.3.4. If the binary was that version, there was a very good chance that we had an easy backdoor that we could use to get root. Unfortunately, on digging into the strings in the binary, we found out that the vsftpd was version 2.3.2 on the router. This meant that there most likely was no backdoor, but on some digging into the version, we discovered that Exploit Database had a dos exploit at EBDID: 16270 [9]. It wasn't exactly what we were looking for, but it was close enough to provide the scrutiny that we were looking for.

*4.2.1 Synopsis of CVE-2011-0762.* CVE-2011-0762 is a Denial of Service(DOS) attack against vsftpd server. It does affects versions 2.3.3 and below. It is a CVE that causes DOS by overusing CPU consumption via specially crafted glob statements, wildcards "?" or



```
root:$1$GTN.gpri$DlSyKvZKMR9A9Uj9e9wR3/:15502:0:99999:7:::
```

**Figure 9**

"*". It causes this overuse due to a memory leak. It affects mainly legitimate users, devices and information systems from accessing to a system.

*4.2.2 Location of Bug.* Once we started debugging and getting more into the code, we have found where the bug or malware is coming from. The failure comes specifically from vsf_filename_passes_filter function inside of the ls.c. There is an excessive memory leak present somewhere in between this function. One of the reasons of this cause are glob expressions in STAT commands in multiple FTP sessions. It has a very intrusive way with backporting the specific kernel patch.

*4.2.3 The Exploit Module.* When using metasploit module, the exploit can be found at *auxiliary/dos/ftp/vstfpd_232* [8]. It has a check to ensure that the target is actually vulnerable. The main issue with it is that it wont work unless multiple instances of the attack are happening at once. Nick attempted to make it work with multithreading, but metasploit has very little documentation. Especially in something more advanced as multithreading in a module. When tested along with the PoC provided by Exploit Database [9], it successfully DOSed the router.

### 4.3 CVE-2015-3035

*4.3.1 Synopsis of CVE-2015-3035.* CVE-2015-3035 is a Directory Traversal attack affecting TP-LINK Archer C5, C7, and C9 routers. This vulnerability allows accessing all files in the filesystem as long as you are aware of that file.

*4.3.2 Location of Bug.* Adversaries use */login/* directory to attack. Attackers manipulate variables referencing files with dot dot slash sequences. Finally, by using these sequences into the file name, adversaries can backtrack up from directory. As a result, the location of bugs is in the directory path.

*4.3.3 The Exploit Module.* When using metasploit module, the exploit can be found at *auxiliary/scanner/http/archer_c7_traversal* [7]. The module does not have a check, but that is mainly because there's no safe way to check besides going for it. We were able to test it and it can consistently and successfully grab files from within the file-system. An additional feature Nick attempted to add to it was a means of supplying multiple files to be checked for a dirbuster like attack, but there wasn't any clean way of doing such that could be found in given example modules.

## 5 DISCUSSION

While there was a lot that we were able to find, there was a good portion that we could not find. Here, we thought we would go over some of the examples of loot that we found but could not find a use for.

### 5.1 Potential uses of found root password

Previously, we stated that the router has a hardcoded password for root. After brutforcing the hash, we found it to be *sohopassword.* From there, we could not find a use for it. The password was not used for logging into the router through its HTTP interface. Though the SSH port was enabled, we could not use our root credentials to log into the SSH port.

The best solution that we could think that the password would be useful will be privilege escalation. The problem with this is that it might not be necessary since most routers run their applications as root out of simplicity. However, if the router uses a second user to prevent full access, these credentials could be used to elevate to root.

### 5.2 Limitations and Future Work

On further inspection, the reader might have noticed that we discussed in depth exploration of the CVE-2011-0762 [3] vulnerability, but we never discussed where the vulnerability for CVE-2015-3035 is located. That is because we do not know. We know that it exists because our exploits work, but we have no idea what causes it. It most likely is a bug with Apache httpd. We checked the binary, but the problem is that apache has config files as well, and they can be in layers. The bug may appear in a config file, or in a *.htaccess* file. We cannot find it because they wouldnt be included in the firmware file. We attempted to use the traversal exploit itself to attempt to dig up its own configs, but that is a blind guessing game.

There was also the SMBD binary that we had. Looking at the Exploit Database, there were some files that we could have checked out. We just didnt. There was a lot that we could have done further. The largest blocker was time. We only had a limited time do work on this project along with other classes. Because of that, this was what we were able to collect. This isn't a complete loss however. For who ever decides to dig further, we have theories on what could cause the vulnerabilities and how to find out.

## 6 WORK DISTRIBUTION

Our group has distributed the workload in three different sections. Nicholas has dedicated hours to three sections but focused mainly in the extracting firmware area section. That been said, he explored and extracted the files needed in order for us to understand and get through it at the moment of finding the bugs that are causing the vulnerabilities. Mana who has also dedicated time to three sections took more responsibility in the searching file-structure area. She worked with Ghindra to understand the actual code behind this files and been able to get through them. And lastly, Anna that also dedicated time to the three sections took the responsibility to researching vulnerabilities. When looking for routers that were available to us to explore and examine if there had any vulnerabilities, she digs into some research to see what CVEs were impactful for the chosen router the group had decided to use. We all three dedicated a significant number of hours that allowed and helped for the contribution of the project.

## 7 CONCLUSION

To describe all that we have accomplished so far: We have managed to extract the Linux Filesystem from the firmware binary that was provided for us to update the router that we are pentesting. We ran into problems while extracting the filesystem, but we managed to fix it utilizing some online resources. From there, we took a look at all common loot that pentesters search for once they manage to access a filesystem, including the versions of programs installed and passwords. We found out that the root password installed from the firmware is not the same as the router's default password, being sohopassword.

Our team was ultimately able to find vulnerabilities on the Archer C7 router. What vulnerabilities we had found, we wrote exploits and submitted them to a public repo for others to use. There were several vulnerabilities that we could not pursue to the fullest extent due to time constraints; however, we left as much information as possible for someone else to investigate further in the future.

## REFERENCES

[1] [n.d.]. *Download for Archer C7: TP-link.* https://www.tp-link.com/us/support/download/archer-c7/v1/#Firmware

[2] [n.d.]. VSFTPD 2.3.4 Download. https://security.appspot.com/downloads/vsftpd-2.3.2.tar.gz

[3] 2011. *CVE-2011-0762.* Retrieved December 3, 2022 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0762

[4] 2011. *Metasploit-Framework: VSFTPD v2.3.4 Backdoor Command Execution.* https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/ftp/vsftpd_234_backdoor.rb

[5] 2015. *CVE-2015-3035.* Retrieved December 3, 2022 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3035

[6] 2020. *Metasploit-Framework: TP-Link Archer A7/C7 Unauthenticated LAN Remote Code Execution.* https://github.com/rapid7/metasploit-framework/blob/master/modules/exploit/linux/misc/tplink_archer_a7_c7_lan_rce.rb

[7] 2022. *Metasploit-Framework: Archer c7 directory traversal.* Retrieved December 3, 2022 from https://github.com/rad10/metasploit-framework/blob/archer_c7_traversal/modules/auxiliary/scanner/http/archer_c7_traversal.rb

[8] 2022. *Metasploit-Framework: VSFTPD DOS attack.* Retrieved December 3, 2022 from https://github.com/rad10/metasploit-framework/blob/vsftpd_232/modules/auxiliary/dos/ftp/vstfpd_232.rb

[9] Maksymilian Arciemowicz. 2011. vsftpd 2.3.2 - Denial of Service. (Mar 2011). Retrieved December 3, 2022 from https://www.exploit-db.com/exploits/16270

[10] Sergio Prado. 2021. Reverse engineering my Router's firmware with binwalk. (2021). https://embeddedbits.org/reverse-engineering-router-firmware-with-binwalk/

[11] Stefan Viehbock. 2015. TP-LINK Local File Disclosure. (Apr 2015). https://packetstormsecurity.com/files/131378/TP-LINK-Local-File-Disclosure.html