

(Reverse Engineering the Firmware Binary)

Extraction, Analysis and Modification of Router Firmware

Contents:

Background-----	Page 2
Objective-----	Page 2
Strategy Followed-----	Page 2
Tools used-----	Page 3
Demonstration-----	Page 3
Mitigation or How to reduce the chance of Exploitation-----	Page 10
Conclusion-----	Page 11

Dilpreet Singh Bajwa

Background: In the world of ever increasing interconnection of computing, mobile and embedded devices, their security has become critical. The security of embedded devices and their firmwares is the new trend in the embedded market. The security requirements and expectations for computing devices are being constantly raised as the world moves towards the Internet of Things. This is especially true for embedded devices and their software counterpart – firmwares – which is also their weakest point.

IoT device Firmware Reverse Engineering: It is a process to understand the device architecture, functionality and vulnerabilities present in the device incorporating different methods.

Firmware: Piece of code written for specific hardware to perform different operations and control the device

Objective: The target device is a router which is running an embedded Linux OS. Routers are the main incoming and outgoing points to the outside world on a computer network, and as such are a prime location for sniffing traffic and performing man in the middle (MITM) attacks. If we control the router then we control the network traffic.

We reverse engineer the firmware binary of a Router, analyse and throw some light that how it can be customised for our purpose.

The purpose is to provide a demonstration that how we can reverse-engineer a firmware binary, extract the file system, pack it again, analyse it and customise it as per our requirement.

Strategy Followed:

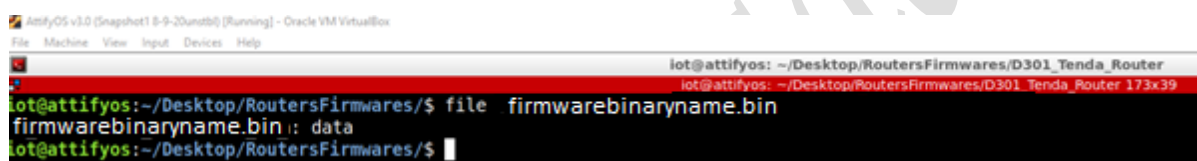
- The firmware for the device can be downloaded from the vendor's website.
- Get as much information about the firmware by using tools like file, readelf, binwalk etc. and from online sources regarding file type, Boot Loader and its version, file system, Architecture, Version, Kernel version, compression used or any encryption used etc.
- File system and kernel can be extracted by using tools like dd, binwalk and firmware-mod-kit.
- Analyse the file system as the file system includes default settings and binaries from the device which then can then be reverse engineered for potential exploits. The purpose of this is to identify the architecture information of the device, as well as finding vulnerabilities and other potential avenues of attack.

Tools Used:

- File tool
- Binwalk
- Readelf
- Firmware-mod-kit
- Firmware-analysis-Tool-Kit

Demonstration of extracting, analysing and customising the firmware:

Figure: 1



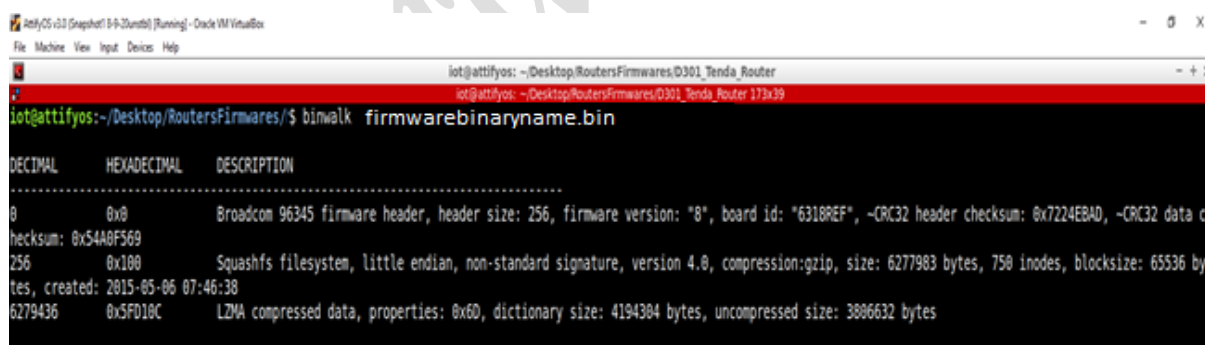
```

iot@attifyos: ~/Desktop/RoutersFirmwares/D301_Tenda_Router
iot@attifyos: ~/Desktop/RoutersFirmwares/D301_Tenda_Router 173x39
iot@attifyos:~/Desktop/RoutersFirmwares/$ file firmwarebinaryname.bin
firmwarebinaryname.bin: data
iot@attifyos:~/Desktop/RoutersFirmwares/$

```

Figure 1 above shows the output of a tool called file, this tool attempts to classify a file and tests if the file is in a certain format. Here it simply tells that the file type is data. We also ran the file tool on an extracted executable from the bin directory in the filesystem later on.

Figure2:



```

iot@attifyos: ~/Desktop/RoutersFirmwares/D301_Tenda_Router
iot@attifyos: ~/Desktop/RoutersFirmwares/D301_Tenda_Router 173x39
iot@attifyos:~/Desktop/RoutersFirmwares/$ binwalk firmwarebinaryname.bin

```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	Broadcom 96345 firmware header, header size: 256, firmware version: "8", board id: "6318REF", -CRC32 header checksum: 0x7224EBAD, -CRC32 data c
256	0x100	Squashfs filesystem, little endian, non-standard signature, version 4.0, compression: gzip, size: 6277983 bytes, 750 inodes, blocksize: 65536 by
6279436	0x5FD10C	LZMA compressed data, properties: 0x6D, dictionary size: 4194384 bytes, uncompressed size: 3886632 bytes

In Figure2 above, we have used “Binwalk” which is an open-source tool for analyzing, reverse engineering and extracting firmware images. Binwalk is able to scan a firmware image and search for file signatures to identify and extract filesystem images, executable code, compressed archives, bootloader and kernel images, file formats like JPEGs and PDFs, and many more!

Both *file* and *binwalk* tools use the *libmagic* library to identify file signatures. But *binwalk* additionally supports a list of custom magic signatures to find compressed/archived files, firmware headers, Linux kernels, bootloaders, filesystems, and so on!

Binwalk has several options you can use but here we only simply use *binwalk* without any option to get useful information about our target firmware image. Binwalk structure has three sections:

1. File location in decimal format
2. File location in hexadecimal format
3. Description of what was found at that location

The image mentions that the firmware belongs to chipset Broadcom 96345 and also firmware version 8 in first line. And based on the image found at the address 0x100, we can see that the rootfs is a squashfs filesystem in little endian format. Other useful information like gzip compression and size is also shown. Further in last line at address 0x5D10C shows that lzma compression is used.

Figure 3:

```

iot@attifyos: ~/tools/firmware-mod-kit$ ls
build-firmware.sh          creating ipkg_packages.htm  extract-firmware.sh        ipkg_remove_all.sh  README.md      uncramfs_all.sh
build-multisquashfs-firmware.sh ddwrt-gui-extract.sh      extract-multisquashfs-firmware.sh ipkg_remove.sh     shared.inc     unsquashfs_all.sh
check_for_upgrade.sh       ddwrt-gui-rebuild.sh      firmware_mod_kit_version.txt  ipkg_template      shared-ng.inc  WikiHackingVendorFirmwares.md
cleanup.sh                 deprecated                ipkg_install_all.sh         other-scripts      src
common.inc                Documentation.md          ipkg_install.sh            ProjectHome.md     uncpio.sh
iot@attifyos:~/tools/firmware-mod-kit$ sudo ./extract-firmware.sh /home/iot/Desktop/RoutersFirmwares/firmwarebinaryname.bin
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Scanning firmware...

Scan Time:      2020-09-13 23:02:16
Target File:    /home/iot/Desktop/RoutersFirmwares/firmwarebinaryname.bin
MD5 Checksum:  724b0290cdd4f3e97bc0613a5f62c1c
Signatures:    344

DECIMAL    HEXADECEMAL  DESCRIPTION
-----
0          0x0          Broadcom 96345 firmware header, header size: 256, firmware version: "8", board id: "6318REF", ~CRC32 header checksum: 0x7224EBAD, ~CRC32 data c
checksum: 0x54A0F569
256        0x100        Squashfs filesystem, little endian, non-standard signature, version 4.0, compression: gzip, size: 6277983 bytes, 750 inodes, blocksize: 65536 by
tes, created: 2015-05-06 07:46:38
6279436    0x5FD10C     LZMA compressed data, properties: 0x60, dictionary size: 4194304 bytes, uncompressed size: 3806632 bytes

Extracting 256 bytes of broadcom header image at offset 0
Extracting squashfs file system at offset 256
Extracting 2848 byte footer from offset 7564413
Extracting squashfs files...
Firmware extraction successful!
Firmware parts can be found in '/home/iot/tools/firmware-mod-kit/fmk/'
iot@attifyos:~/tools/firmware-mod-kit$

```

Now we can extract file system from the firmware image by either using *dd* command or by automatic extracting by using *binwalk* *-e* option or by using *firmware-mod kit*. Here in above figure we have used *firmware-mod-kit* as we also need to build it again after modification and in that case *firmware-mod-kit* provides us that functionality.

We extract the firmware by using ./extract-firmware.sh script of firmware-mod-kit and able to extract filesystem successfully. The firmware-mod-kit create a new directory “fmk” and store all extracted files in it.

Figure 4:

```

AttifyOS v3.0 (Snapshot 8-9-20unstable) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

iot@attifyos: ~/tools/firmware-mod-kit/fmk/rootfs
iot@attifyos: ~/tools/firmware-mod-kit/fmk/rootfs 173x39

iot@attifyos:~/tools/firmware-mod-kit$ cd fmk
iot@attifyos:~/tools/firmware-mod-kit/fmk$ ls
image_parts  logs  rootfs
iot@attifyos:~/tools/firmware-mod-kit/fmk$ cd rootfs/
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$ ls
bin  data  dev  etc  lib  linuxrc  mnt  net  proc  sbin  sys  tmp  usr  var  webs
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$

```

As shown above in figure 4, rootfs directory contains the complete filesystem of our firmware image which we can further analyse to find vulnerabilities and how we can modify it in our best interest.

Figure 5:

```

iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$ ls
bin  data  dev  etc  lib  linuxrc  mnt  net  proc  sbin  sys  tmp  usr  var  webs
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$ cd bin
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/bin$ ls
acs_cli  brctl  ddnsd  dhcpc  ebtables  find_idx  iptables  mkdir  nvrampUpdate  radvd  smbpasswd  tendaIPRangeBandwidth  vlantctl  xdsctl
acsd  busybox  deluser  dhcpd  echo  grep  kill  mknode  openssl  rastatus6  snd  tendaupload  wl  xtm
adsl  cat  detectEth  dnsmasq  epi_tftp  hotplug  lld2d  mount  ping  snmpd  tr6pc  wlctl  xtctl
adslctl  chmod  detectPVC  dnsmasq  ethswctl  hspatp  ln  mroute  ping6  rra  snmp  true  wlevt
ash  collectvcc  detectWan  dnsspoof  ethswctl  httpd  ls  nas  pppd  setmem  ssk  udhcpd  wlmngr
ate  console  df  dsdiagd  false  ip  mcp  nas4not  probevcc  sh  stty  umount  wl_server
bash  cp  dhcp6c  dumpmem  fc  ip6tables  mcpctl  ntfs-3g  ps  sleep  tc  upnp  wl_server_socket
bcmserver  date  dhcp6s  eapd  fcctl  ipdd  mcpd  nvramp  pwd  smb  telnetd  urlfilterd  wps_monitor
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/bin$ readelf -a busybox | more
ELF Header:
  Magic:   7f 45 4c 01 02 01 00 00 00 00 00 00 00 00 00 00
  Class:   ELF32
  Data:    2's complement, big endian
  Version: 1 (current)
  OS/ABI:  UNIX - System V
  ABI Version:
  Type:    EXEC (Executable file)
  Machine: MIPS R3000
  Version: 0x1
  Entry point address: 0x404030
  Start of program headers: 52 (bytes into file)
  Start of section headers: 0 (bytes into file)
  Flags:    0x50001007, noorder, pic, cpic, o32, mips32
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 7
  Size of section headers: 0 (bytes)
  Number of section headers: 0
  Section header string table index: 0

```

Now as shown in Figure 5, Figure 6 and Figure 7, we traverse various directories like bin, etc and webs which contains important files which we further analyse to find any loopholes which either helps us in creating any backdoor, bypass authentication or we can modify to brick the device. For example, there are some start-up scripts present in some directories which runs on start-up of the router and do various work like initialization and setting various parameters. We may modify such files to achieve our objective. (The name and location of such files is not mentioned here intentionally).

Figure 6:

```

iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/bin$ cd ..
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$ ls
bin data dev linuxrc mnt proc sys tmp var
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs$ cd webs
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/webs$ ls
accessCtrl.html D1201.png dsladderr.html html5shiv.js menuTitle.js quicksetuptestsucc.html statsadslerr.html upnpconf.html
adslcfadv.html D1201usb.gif dslbondingcfg.html icon-easy.png multicast.html reasy.css statsadsl.html url_add.html
adslcfadv.html D301.gif enblbridge.html ifcdns.html reasy-ui.js statsadslreset.html statsadsl.html usb.gif
adslcfadv.html ddnsadd.html enblservice.html ifcgateway.html rebootinfo.html statsatmerr.html statsatm.html util.js
adslcfadv.html defaultsettings.html engdebug.html index.html ntwksum2.html resetroute.html statsatmreset.html statsatm.html wanaddderr.html
adv_conf.gif deng1_302.gif ethadderr.html info.html password.html routeadd.html statsatmreset.html statsatm.html wanaddderr.html
aigcfadv.html deng1_302.gif footer.html ipocfadv.html portmapadd.html rtdefaultcfadv.html statsatmreset.html statsatm.html wanaddderr.html
backuptest.html deng2.gif g3cfadv.html ipocfadv.html portmapadd.html rtdefaultcfadv.html statsatmreset.html statsatm.html wanaddderr.html
berun.html deng2.gif g3cfadv.html ipocfadv.html portmapadd.html rtdefaultcfadv.html statsatmreset.html statsatm.html wanaddderr.html
berstart.html deng2.gif g3cfadv.html ipocfadv.html portmapadd.html rtdefaultcfadv.html statsatmreset.html statsatm.html wanaddderr.html
berstop.html diag0802lag.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
blank3g.html diag.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
certadd.html diag.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
certcaimport.html diagipow.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
certimport.html diagipow.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
certloadsigned.html diagipow.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
cfadv.html diagipow.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
countrycode.js dnscfg.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
D1201deng1.gif dnscfg.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
D1201deng3.gif dnscfg.html hlpaddsync.html j.js pradd.html pmwngmt.html qoscfadv.html qoscl.html statsatmreset.html statsatm.html wanaddderr.html
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/webs$

```

Figure 7:

```

iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$ ls
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$ cd etc
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc$ ls
adsl dhcp6c.conf.sample fstab HP_P1008.img ipsec.conf passwd racoon.conf smb.conf udhcpd.leases
arl dns.conf gateway.conf inetd.conf ipv6_start.sample mdk radvd.conf.sample soft bridge vlan
default.cfg dyndsc.sh group initab modules_install profile samba sysmsg wrt54g.large.ico
dhcp6c.conf.sample filesystems hotplug_add_usbld_and_hp_p1008 initab modules_install profile samba sysmsg wrt54g.large.ico
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc$ cd init.d/
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$ ls
rcS
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$

```

Figure 8:

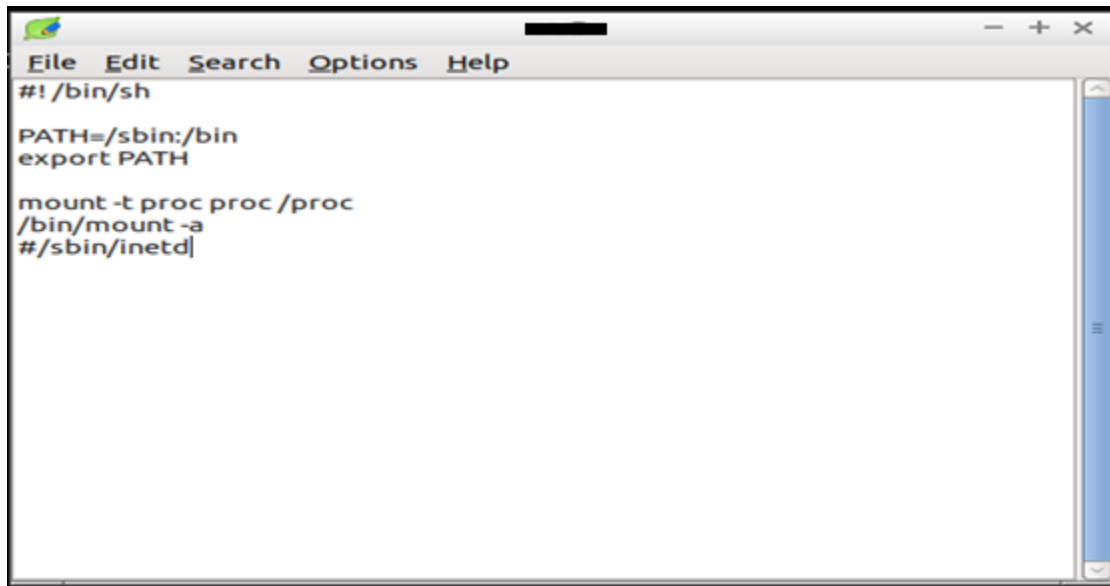
```

iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$ ls
rcS
iot@attifyos:~/tools/firmware-mod-kit/fmk/rootfs/etc/init.d$ leafpad startuptscriptfilename

```

As shown above in Figure 8, we simply accessing the start-up script file and open it withleafpad. The contents of file are shown below in Figure 9.

Figure 9:



As we know from above discussion that this file is important and runs on router start-up and we also knew that the system is a Linux system. We further analyse busybox file present in /bin directory to know its version, commands it supports by using file and emulation software qemu. We can as per our requirement can write our customised code or script and can run through it on system start-up. It is just one method, there are number of files available and we further explore them as per our need. Our malicious process will run in background.

Figure 11:

```

iot@attifyos:~/tools/firmware-mod-kit$ sudo ./build-firmware.sh
[sudo] password for iot:
Firmware Mod Kit (build) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Building new squashfs file system... (this may take several minutes!)
Squashfs block size is 64 Kb
Parallel mksquashfs: Using 1 processor
Creating 4.0 filesystem on /home/iot/tools/firmware-mod-kit/fmk/new-file-system.squashfs, block size 65536.
=====
Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 65536
  compressed data, compressed metadata, compressed fragments, compressed xattrs
  duplicates are removed
Filesystem size 7716.35 Kbytes (7.54 Mbytes)
  35.85% of uncompressed filesystem size (21521.43 Kbytes)
Inode table size 7317 bytes (7.15 Kbytes)
  38.42% of uncompressed inode table size (24050 bytes)
Directory table size 7208 bytes (7.04 Kbytes)
  54.73% of uncompressed directory table size (13170 bytes)
Number of duplicate files found 1
Number of inodes 750
Number of files 397
Number of fragments 63
Number of symbolic links 103
Number of device nodes 203
Number of fifo nodes 1
Number of socket nodes 0
Number of directories 46
Number of ids (unique uids + gids) 1
  Number of uids 1
    root (0)
  Number of gids 1
    root (0)
  
```

After Modifying the required files, we simply build our new modified firmware again by using firmware-mod-kit as shown in figure 11 above by using script ./build-firmware.sh available in firmware-mod-kit and the new-firmware file is also stored in fmk directory with name new-firmware.bin as shown below in Figure12.

Figure 12:



```

AttifyOS v3.0 (Snapshot1 8-9-20unstbl) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

iot@attifyos: ~/tools/firmware-mod-kit/fmk
iot@attifyos: ~/tools/firmware-mod-kit/fmk 173x39

iot@attifyos:~/tools/firmware-mod-kit/fmk$ ls
image_parts  logs  new-filesystem.squashfs  new-firmware.bin  rootfs
iot@attifyos:~/tools/firmware-mod-kit/fmk$

```

Figure 13:



```

AttifyOS v3.0 (Snapshot1 8-9-20unstbl) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

iot@attifyos: ~/tools/firmware-analysis-toolkit
iot@attifyos: ~/tools/firmware-analysis-toolkit 173x39

iot@attifyos:~/tools$ cd firmware-analysis-toolkit/
iot@attifyos:~/tools/firmware-analysis-toolkit$ sudo ./fat.py /home/iot/Desktop/RoutersFirmwares/Originalfirmwarebinaryname.bin

      Fat
  _____
 Welcome to the Firmware Analysis Toolkit - v0.3
 Offensive IoT Exploitation Training http://bit.do/offensiveiotexploitation
 By Attify - https://attify.com | @attifyme

[+] Firmware: originalfirmwarebinaryname.bin
[+] Extracting the firmware...
[+] Image ID: 23
[+] Identifying architecture...
[+] Architecture: mips64
[+] Building QEMU disk image...
[+] Setting up the network connection, please standby...
[+] Network interfaces: []
[+] All set! Press ENTER to run the firmware...
[+] When running, press Ctrl + A X to terminate qemu.

```

We modify or customised the firmware binary in a way that after uploading our new customised binary to the router, the router become unusable that it does not work. Here we emulate it through firmware-analysis-tool kit which is based on qemu and automate the whole process.

First we emulate our original firmware file as shown in Figure 13 and Figure 14 and finds that it works fine. We able to emulate it and also accessing its file system (Figure 14). This confirms us that the original firmware is working fine.

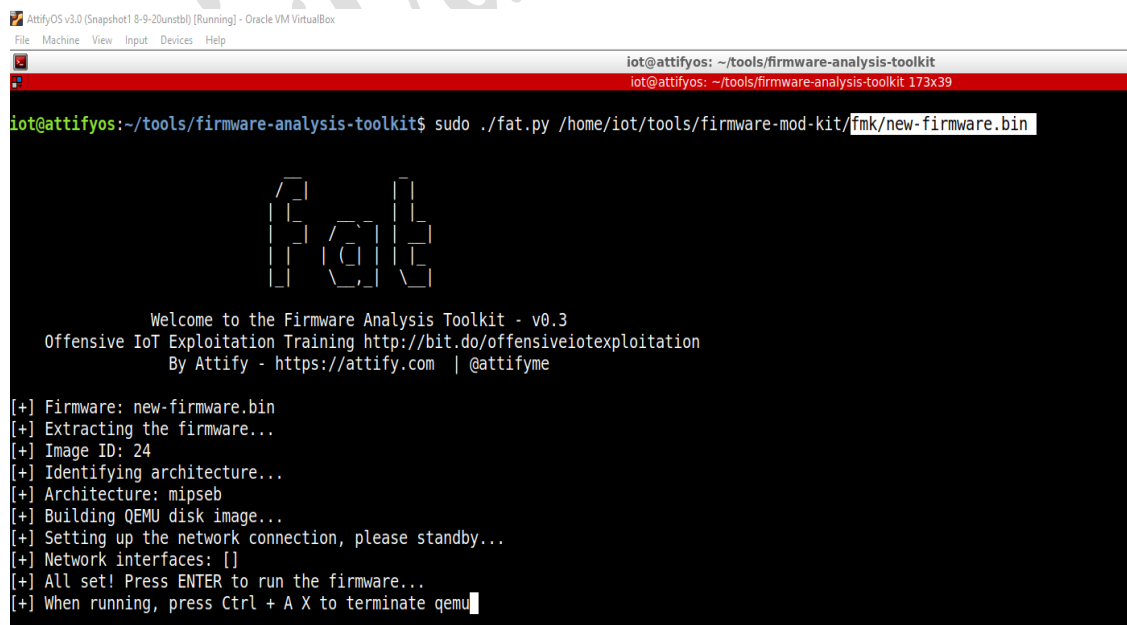
Now our next task is to emulate our modified new-firmware and check whether we can be able to brick the router or not with our modification.

Figure 14:

```
# pwd
/
# ls
bin      etc      linuxrc  opt      sys      var
data     firmadyne lost+found proc      tmp      webs
dev      lib      mnt      sbin     usr

# cd bin
# ls
acs_cli      ethswctl      radvd
acsd         false         rastatus6
adsl         fc            rawSocketTest
adslctl      fcctl         rm
ash          find_idx      setmem
ate          grep          sh
bash         hotplug       sleep
bcmmserver   hspotap       smbd
brctl        httpd         smbpasswd
busybox      ip            smd
cat          iptables     snmpd
chmod        ipdd         snmp
collectvcc   iptables     ssk
consoled     kill         stty
cp           lld2d        tc
date         ln            telnetd
ddnsd        ls            tendaIPRangeBandwidth
deluser      mcp          tendaupload
detectEth    mcpctl       tr69c
detectPVC    mcpd         true
detectWan    mkdir        udhcpd
df           mknod        umount
```

Figure 15:



```
iot@attifyos: ~/tools/firmware-analysis-toolkit
iot@attifyos: ~/tools/firmware-analysis-toolkit 173x39

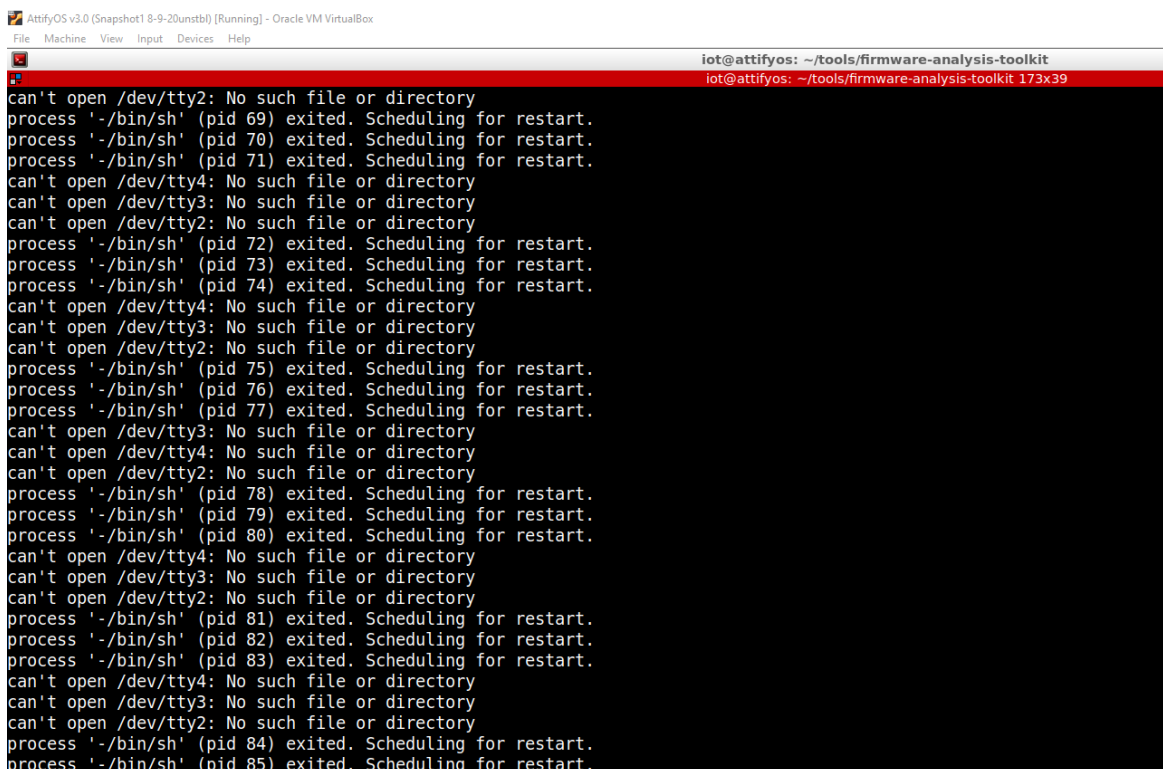
iot@attifyos:~/tools/firmware-analysis-toolkit$ sudo ./fat.py /home/iot/tools/firmware-mod-kit/fmk/new-firmware.bin

      f a t
    Welcome to the Firmware Analysis Toolkit - v0.3
    Offensive IoT Exploitation Training http://bit.do/offensiveiotexploitation
    By Attify - https://attify.com | @attifyme

[+] Firmware: new-firmware.bin
[+] Extracting the firmware...
[+] Image ID: 24
[+] Identifying architecture...
[+] Architecture: mipseb
[+] Building QEMU disk image...
[+] Setting up the network connection, please standby...
[+] Network interfaces: []
[+] All set! Press ENTER to run the firmware...
[+] When running, press Ctrl + A X to terminate qemu
```

As shown in Figure 15, we are now emulating our modified new-firmware file with firmware-analysis-tool-kit. As shown in Figure 16 below that emulation is failed because of our modification in the firmware and router is restarting again and again. Our purpose and objective is fulfilled.

Figure 16:



```

AttifyOS v3.0 (Snapshot1 8-9-20unstable) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

iot@attifyos: ~/tools/firmware-analysis-toolkit
iot@attifyos: ~/tools/firmware-analysis-toolkit 173x39

can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 69) exited. Scheduling for restart.
process './bin/sh' (pid 70) exited. Scheduling for restart.
process './bin/sh' (pid 71) exited. Scheduling for restart.
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 72) exited. Scheduling for restart.
process './bin/sh' (pid 73) exited. Scheduling for restart.
process './bin/sh' (pid 74) exited. Scheduling for restart.
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 75) exited. Scheduling for restart.
process './bin/sh' (pid 76) exited. Scheduling for restart.
process './bin/sh' (pid 77) exited. Scheduling for restart.
can't open /dev/tty3: No such file or directory
can't open /dev/tty4: No such file or directory
can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 78) exited. Scheduling for restart.
process './bin/sh' (pid 79) exited. Scheduling for restart.
process './bin/sh' (pid 80) exited. Scheduling for restart.
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 81) exited. Scheduling for restart.
process './bin/sh' (pid 82) exited. Scheduling for restart.
process './bin/sh' (pid 83) exited. Scheduling for restart.
can't open /dev/tty4: No such file or directory
can't open /dev/tty3: No such file or directory
can't open /dev/tty2: No such file or directory
process './bin/sh' (pid 84) exited. Scheduling for restart.
process './bin/sh' (pid 85) exited. Scheduling for restart.

```

Mitigation or How to reduce the chance of Exploitation:

Vendors can perform a few simple steps to make life for the attacker a lot harder than it currently is:-

- Remove all debugging ports (serial, JTAG)
- Remove all unnecessary services such as telnetd
- Audit and lockdown the web interface
- Filter all input into the device.
- Encrypt and Sign firmware.

These steps, especially encrypting the firmware, make offline analysis difficult unless the encryption keys are leaked.

Conclusion:

Here, we have analysed, extracted the entire filesystem of the target Firmware. We further modified the firmware and check through emulation that it did not work after modification. Here we cover only main aspects for firmware extraction, analysis and modification.

Note: Here in the report, the original name of firmware binary, its version, firmware vendor name and start-up scripts files name and location are not mentioned intentionally for the purpose.

BAJWA Academy