Firmware Analysis of Embedded Systems

Dimitrios-Georgios Akestoridis

Carnegie Mellon University

14-829 / 18-638: Mobile and IoT Security (Fall 2018)

Reminders

- University Policies: https://www.cmu.edu/policies/index.html
- Course Policies: http://mews.sv.cmu.edu/teaching/14829/f18/policy.html
- Be aware of potential ethical and legal implications of your actions
- Use isolated networks for your assignments and research

What is an embedded system?

- An embedded system consists of **special-purpose** computer hardware and software, often as part of a larger system and with limited resources

- Embedded systems can be found in a plethora of devices, including:
  - Thermostats
  - Washing machines
  - Pacemakers

- Most IoT devices are just embedded systems with networking capabilities, such as:
  - IP cameras
  - Fitness trackers
  - Smart locks

How do embedded systems work?

- The special-purpose computer software that controls an embedded system is often referred to as **firmware** and it is stored in non-volatile memory

How do embedded systems work?

- The special-purpose computer software that controls an embedded system is often referred to as **firmware** and it is stored in non-volatile memory
- Many vendors use flash memory in their devices to store their firmware, which enables them to later:
    - Improve the system’s functionality
    - Fix security vulnerabilities

How do embedded systems work?

- The special-purpose computer software that controls an embedded system is often referred to as **firmware** and it is stored in non-volatile memory
- Many vendors use flash memory in their devices to store their firmware, which enables them to later:
  - Improve the system’s functionality
  - Fix security vulnerabilities
- A **firmware image** may be provided in order to update the firmware of a device, which can be done either manually or automatically

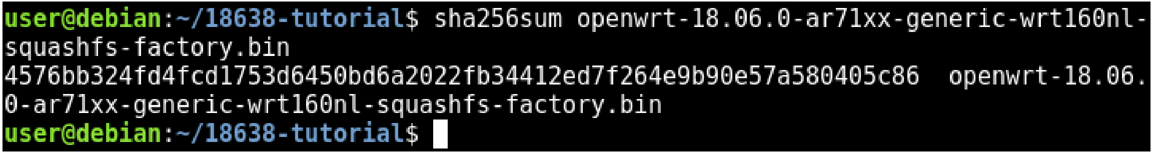
What does a firmware image look like?

- Possible methods for obtaining the firmware image of a device:
  - Downloading it from the vendor’s website
  - Capturing it during the device’s firmware update process
  - Extracting it from the hardware

What does a firmware image look like?

- Possible methods for obtaining the firmware image of a device:
  - Downloading it from the vendor’s website
  - Capturing it during the device’s firmware update process
  - Extracting it from the hardware

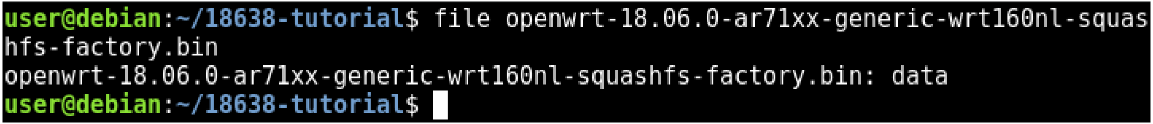
- For illustration purposes, we will use a firmware image from the OpenWrt project:
  - https://downloads.openwrt.org/releases/18.06.0/targets/ar71xx/generic/
  - https://git.openwrt.org/openwrt/openwrt.git/



```
user@debian:~/18638-tutorial$ sha256sum openwrt-18.06.0-ar71xx-generic-wrt160nl-
squashfs-factory.bin
4576bb324fd4fcd1753d6450bd6a2022fb34412ed7f264e9b90e57a580405c86  openwrt-18.06.
0-ar71xx-generic-wrt160nl-squashfs-factory.bin
user@debian:~/18638-tutorial$ 
```

$ file

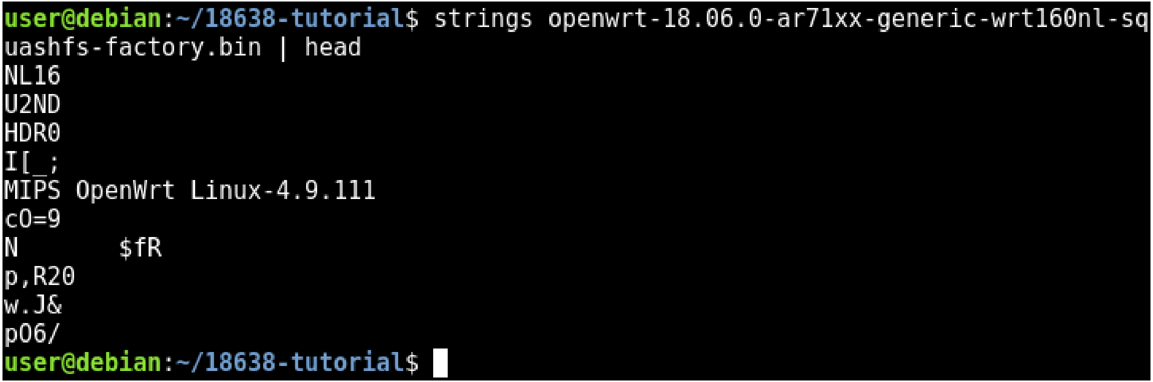
- The firmware image could be in a standard archive format that the `file` command can identify
- If the file format of the provided firmware image is unknown, then `file` will simply report that it contains binary data



```
user@debian:~/18638-tutorial$ file openwrt-18.06.0-ar71xx-generic-wrt160nl-squas
hfs-factory.bin
openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin: data
user@debian:~/18638-tutorial$ 
```

$ strings

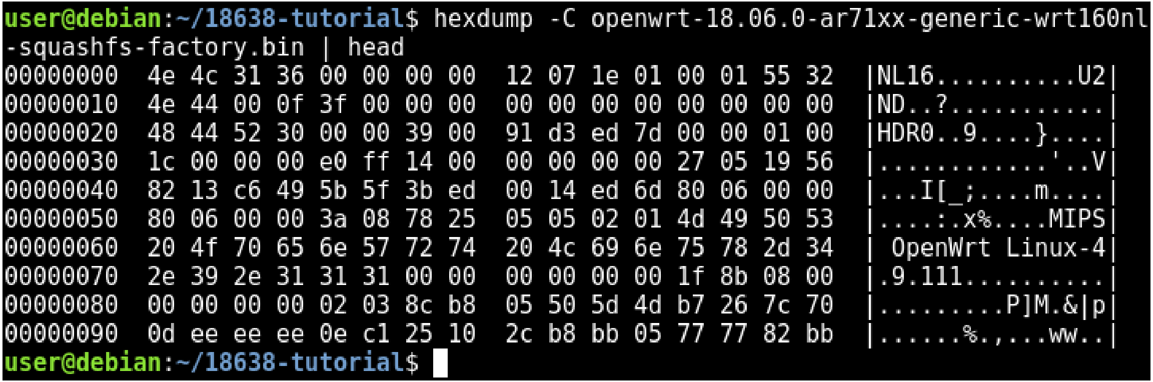
- We can inspect sequences of printable characters in the firmware image with the `strings` command



```
user@debian:~/18638-tutorial$ strings openwrt-18.06.0-ar71xx-generic-wrt160nl-sq
uashfs-factory.bin | head
NL16
U2ND
HDR0
I[_;
MIPS OpenWrt Linux-4.9.111
c0=9
N      $fR
p,R20
w.J&
p06/
user@debian:~/18638-tutorial$ 
```

$ hexdump

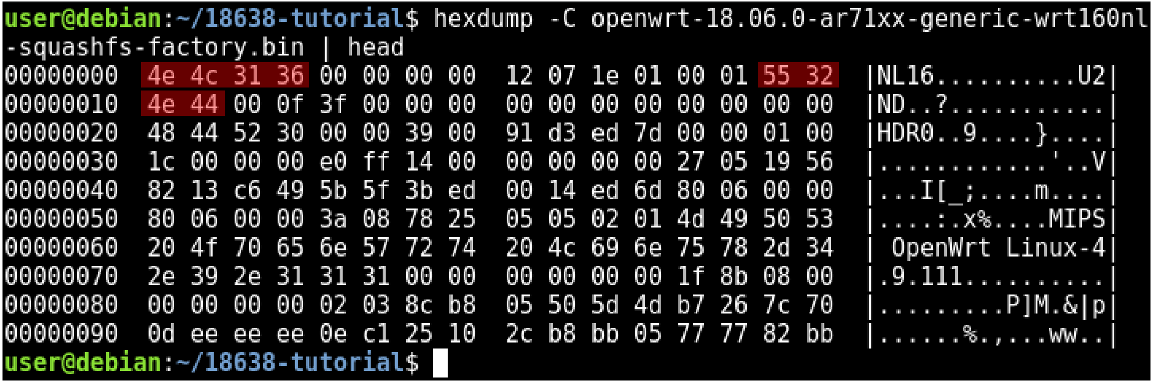
- We can examine the bytes of the firmware image with the hexdump command



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | head
00000000  4e 4c 31 36 00 00 00 00  12 07 1e 01 00 01 55 32  |NL16..........U2|
00000010  4e 44 00 0f 3f 00 00 00  00 00 00 00 00 00 00 00  |ND..?...........|
00000020  48 44 52 30 00 00 39 00  91 d3 ed 7d 00 00 01 00  |HDR0..9....}....|
00000030  1c 00 00 00 e0 ff 14 00  00 00 00 00 27 05 19 56  |............'..V|
00000040  82 13 c6 49 5b 5f 3b ed  00 14 ed 6d 80 06 00 00  |...I[_;....m....|
00000050  80 06 00 00 3a 08 78 25  05 05 02 01 4d 49 50 53  |....:.x%....MIPS|
00000060  20 4f 70 65 6e 57 72 74  20 4c 69 6e 75 78 2d 34  | OpenWrt Linux-4|
00000070  2e 39 2e 31 31 31 00 00  00 00 00 00 1f 8b 08 00  |.9.111..........|
00000080  00 00 00 00 02 03 8c b8  05 50 5d 4d b7 26 7c 70  |.........P]M.&|p|
00000090  0d ee ee ee 0e c1 25 10  2c b8 bb 05 77 77 82 bb  |......%.,...ww..|
user@debian:~/18638-tutorial$
```

$ hexdump

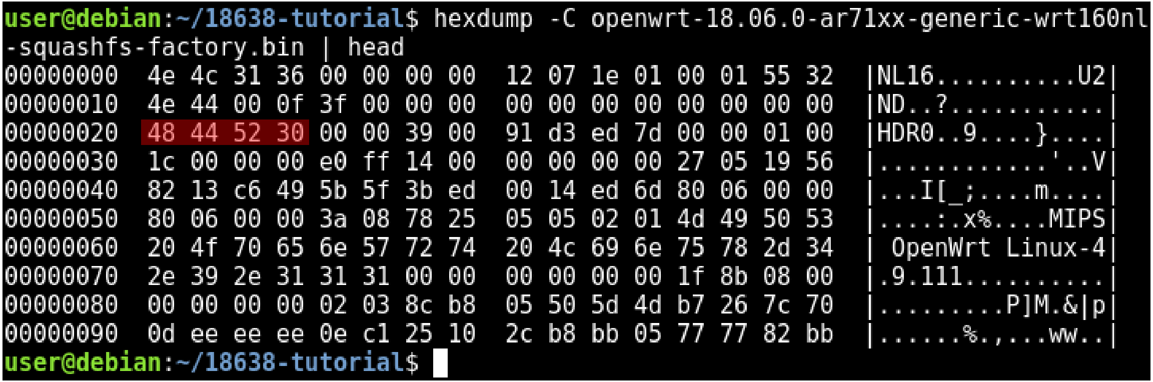
- 0x4e4c3136 (NL16) and 0x55324e44 (U2ND) correspond to the magic number and ID number of the BIN header:
  - https://git.openwrt.org/?p=openwrt/openwrt.git;a=blob_plain;f=tools/firmware-utils/src/addpattern.c;hb=HEAD



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | head
00000000  4e 4c 31 36 00 00 00 00  12 07 1e 01 00 01 55 32  |NL16..........U2|
00000010  4e 44 00 0f 3f 00 00 00  00 00 00 00 00 00 00 00  |ND..?...........|
00000020  48 44 52 30 00 00 39 00  91 d3 ed 7d 00 00 01 00  |HDR0..9....}....|
00000030  1c 00 00 00 e0 ff 14 00  00 00 00 00 27 05 19 56  |............'..V|
00000040  82 13 c6 49 5b 5f 3b ed  00 14 ed 6d 80 06 00 00  |...I[_;....m....|
00000050  80 06 00 00 3a 08 78 25  05 05 02 01 4d 49 50 53  |....:.x%....MIPS|
00000060  20 4f 70 65 6e 57 72 74  20 4c 69 6e 75 78 2d 34  | OpenWrt Linux-4|
00000070  2e 39 2e 31 31 31 00 00  00 00 00 00 1f 8b 08 00  |.9.111..........|
00000080  00 00 00 00 02 03 8c b8  05 50 5d 4d b7 26 7c 70  |.........P]M.&|p|
00000090  0d ee ee ee 0e c1 25 10  2c b8 bb 05 77 77 82 bb  |......%.,...ww..|
user@debian:~/18638-tutorial$ 
```

$ hexdump

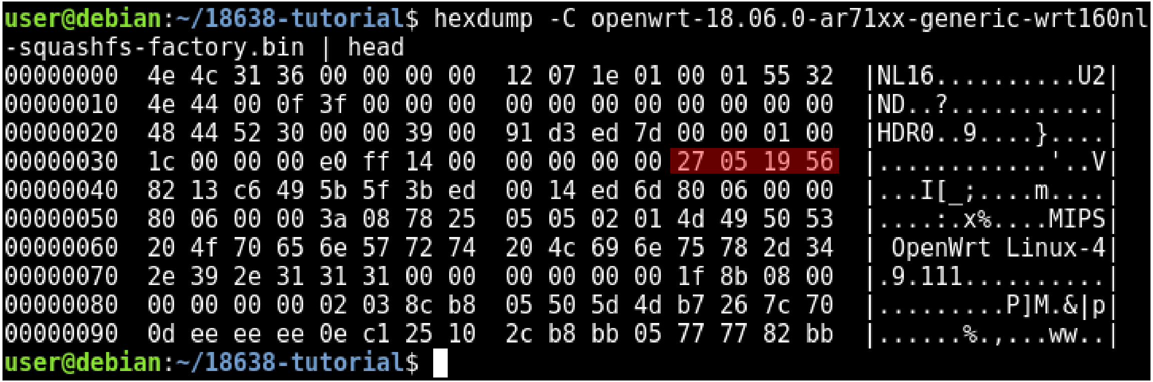
- `0x48445230` (HDR0) corresponds to the magic number of the TRX header:
    - https://git.openwrt.org/?p=openwrt/openwrt.git;a=blob_plain;f=package/system/mtd/src/trx.c;hb=HEAD



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | head
00000000  4e 4c 31 36 00 00 00 00  12 07 1e 01 00 01 55 32  |NL16..........U2|
00000010  4e 44 00 0f 3f 00 00 00  00 00 00 00 00 00 00 00  |ND..?...........|
00000020  48 44 52 30 00 00 39 00  91 d3 ed 7d 00 00 01 00  |HDR0..9....}....|
00000030  1c 00 00 00 e0 ff 14 00  00 00 00 00 27 05 19 56  |............'..V|
00000040  82 13 c6 49 5b 5f 3b ed  00 14 ed 6d 80 06 00 00  |...I[_;....m....|
00000050  80 06 00 00 3a 08 78 25  05 05 02 01 4d 49 50 53  |....:.x%....MIPS|
00000060  20 4f 70 65 6e 57 72 74  20 4c 69 6e 75 78 2d 34  | OpenWrt Linux-4|
00000070  2e 39 2e 31 31 31 00 00  00 00 00 00 1f 8b 08 00  |.9.111..........|
00000080  00 00 00 00 02 03 8c b8  05 50 5d 4d b7 26 7c 70  |.........P]M.&|p|
00000090  0d ee ee ee 0e c1 25 10  2c b8 bb 05 77 77 82 bb  |......%.,...ww..|
user@debian:~/18638-tutorial$ 
```

$ hexdump

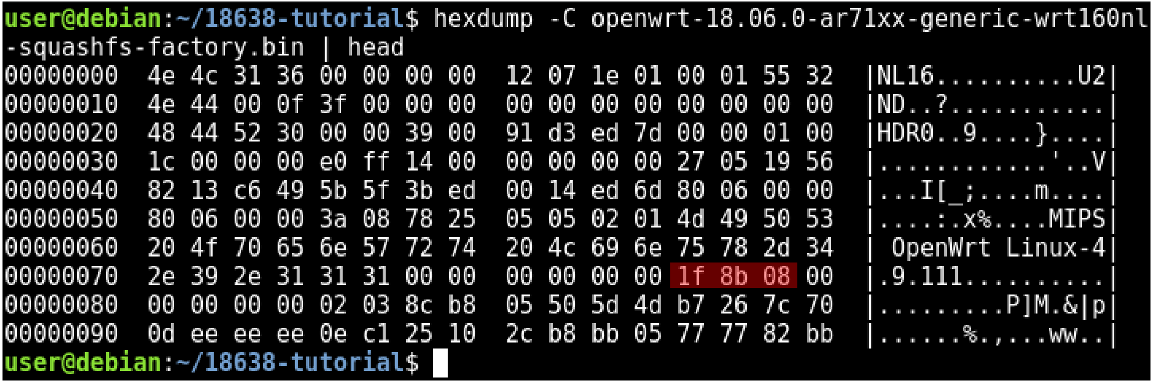
- 0x27051956 corresponds to the magic number of the uImage header:
    - https://git.denx.de/?p=u-boot.git;a=blob_plain;f=include/image.h;hb=HEAD



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | head
00000000  4e 4c 31 36 00 00 00 00  12 07 1e 01 00 01 55 32  |NL16..........U2|
00000010  4e 44 00 0f 3f 00 00 00  00 00 00 00 00 00 00 00  |ND..?...........|
00000020  48 44 52 30 00 00 39 00  91 d3 ed 7d 00 00 01 00  |HDR0..9....}....|
00000030  1c 00 00 00 e0 ff 14 00  00 00 00 00 27 05 19 56  |............'..V|
00000040  82 13 c6 49 5b 5f 3b ed  00 14 ed 6d 80 06 00 00  |...I[_;....m....|
00000050  80 06 00 00 3a 08 78 25  05 05 02 01 4d 49 50 53  |....:.x%....MIPS|
00000060  20 4f 70 65 6e 57 72 74  20 4c 69 6e 75 78 2d 34  | OpenWrt Linux-4|
00000070  2e 39 2e 31 31 31 00 00  00 00 00 00 1f 8b 08 00  |.9.111..........|
00000080  00 00 00 00 02 03 8c b8  05 50 5d 4d b7 26 7c 70  |.........P]M.&|p|
00000090  0d ee ee ee 0e c1 25 10  2c b8 bb 05 77 77 82 bb  |......%.,...ww..|
user@debian:~/18638-tutorial$
```

$ hexdump

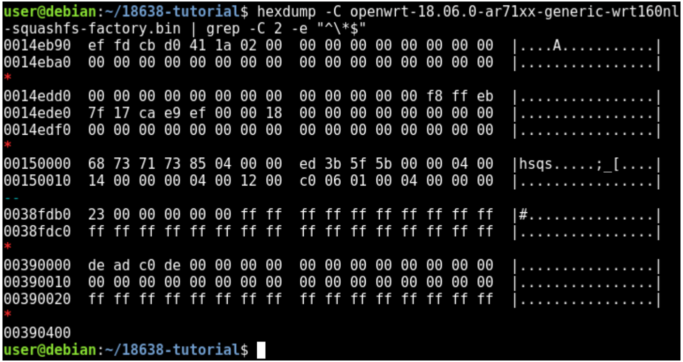
- `0x1f8b08` corresponds to the magic number of the gzip file format with the “deflate” compression method:
  - https://tools.ietf.org/html/rfc1952



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | head
00000000  4e 4c 31 36 00 00 00 00  12 07 1e 01 00 01 55 32  |NL16..........U2|
00000010  4e 44 00 0f 3f 00 00 00  00 00 00 00 00 00 00 00  |ND..?...........|
00000020  48 44 52 30 00 00 39 00  91 d3 ed 7d 00 00 01 00  |HDR0..9....}....|
00000030  1c 00 00 00 e0 ff 14 00  00 00 00 00 27 05 19 56  |............'..V|
00000040  82 13 c6 49 5b 5f 3b ed  00 14 ed 6d 80 06 00 00  |...I[_;....m....|
00000050  80 06 00 00 3a 08 78 25  05 05 02 01 4d 49 50 53  |....:.x%....MIPS|
00000060  20 4f 70 65 6e 57 72 74  20 4c 69 6e 75 78 2d 34  | OpenWrt Linux-4|
00000070  2e 39 2e 31 31 31 00 00  00 00 00 00 1f 8b 08 00  |.9.111..........|
00000080  00 00 00 00 02 03 8c b8  05 50 5d 4d b7 26 7c 70  |.........P]M.&|p|
00000090  0d ee ee ee 0e c1 25 10  2c b8 bb 05 77 77 82 bb  |......%.,...ww..|
user@debian:~/18638-tutorial$ 
```

$ hexdump

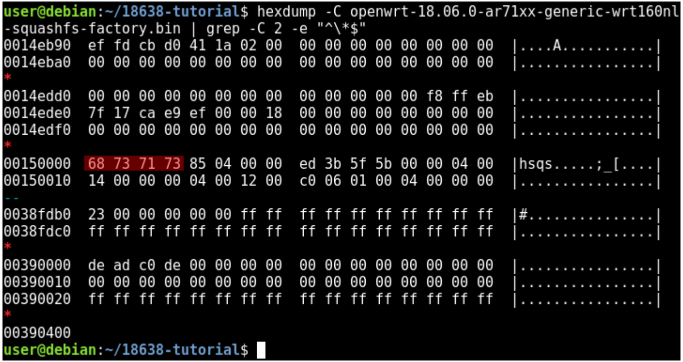
- If the -v option is not provided, hexdump replaces repeating lines with a single asterisk (*)



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | grep -C 2 -e "^\*$"
0014eb90  ef fd cb d0 41 1a 02 00  00 00 00 00 00 00 00 00  |....A...........|
0014eba0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0014edd0  00 00 00 00 00 00 00 00  00 00 00 00 00 f8 ff eb  |................|
0014ede0  7f 17 ca e9 ef 00 00 18  00 00 00 00 00 00 00 00  |................|
0014edf0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00150000  68 73 71 73 85 04 00 00  ed 3b 5f 5b 00 00 04 00  |hsqs.....;_[....|
00150010  14 00 00 00 04 00 12 00  c0 06 01 00 04 00 00 00  |................|
--
0038fdb0  23 00 00 00 00 00 ff ff  ff ff ff ff ff ff ff ff  |#...............|
0038fdc0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390000  de ad c0 de 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390400
user@debian:~/18638-tutorial$
```

$ hexdump

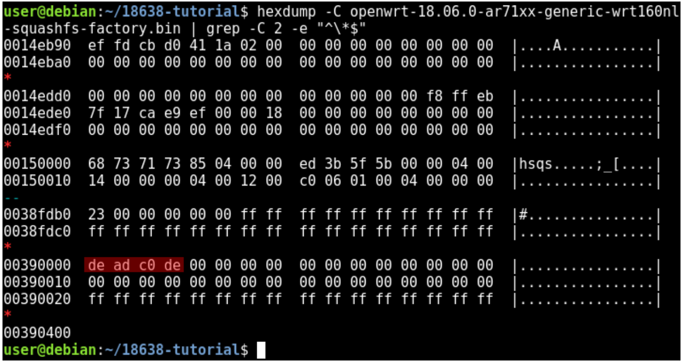
- 0x68737173 (hsqs) corresponds to the magic number of the little-endian SquashFS filesystem:
    - https://sourceforge.net/p/squashfs/code/ci/master/tree/squashfs-tools/squashfs_fs.h



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | grep -C 2 -e "^\*$"
0014eb90  ef fd cb d0 41 1a 02 00  00 00 00 00 00 00 00 00  |....A...........|
0014eba0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0014edd0  00 00 00 00 00 00 00 00  00 00 00 00 00 f8 ff eb  |................|
0014ede0  7f 17 ca e9 ef 00 00 18  00 00 00 00 00 00 00 00  |................|
0014edf0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00150000  68 73 71 73 85 04 00 00  ed 3b 5f 5b 00 00 04 00  |hsqs.....;_[....|
00150010  14 00 00 00 04 00 12 00  c0 06 01 00 04 00 00 00  |................|
--
0038fdb0  23 00 00 00 00 00 ff ff  ff ff ff ff ff ff ff ff  |#...............|
0038fdc0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390000  de ad c0 de 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390400
user@debian:~/18638-tutorial$
```

$ hexdump

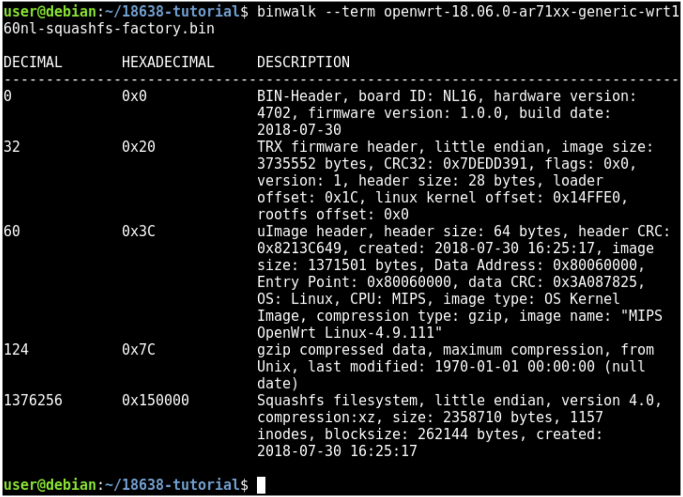
- `0xdeadc0de` indicates the start of the reformatted JFFS2 partition:
    - https://openwrt.org/docs/techref/filesystems



```
user@debian:~/18638-tutorial$ hexdump -C openwrt-18.06.0-ar71xx-generic-wrt160nl
-squashfs-factory.bin | grep -C 2 -e "^\*$"
0014eb90  ef fd cb d0 41 1a 02 00  00 00 00 00 00 00 00 00  |....A...........|
0014eba0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0014edd0  00 00 00 00 00 00 00 00  00 00 00 00 00 f8 ff eb  |................|
0014ede0  7f 17 ca e9 ef 00 00 18  00 00 00 00 00 00 00 00  |................|
0014edf0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00150000  68 73 71 73 85 04 00 00  ed 3b 5f 5b 00 00 04 00  |hsqs.....;_[....|
00150010  14 00 00 00 04 00 12 00  c0 06 01 00 04 00 00 00  |................|
--
0038fdb0  23 00 00 00 00 00 ff ff  ff ff ff ff ff ff ff ff  |#...............|
0038fdc0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390000  de ad c0 de 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00390020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00390400
user@debian:~/18638-tutorial$
```

$ binwalk

- We can use `binwalk` to scan for known signatures
- Custom signatures can easily be incorporated
- Wide variety of analysis options available
- https://github.com/ReFirmLabs/binwalk



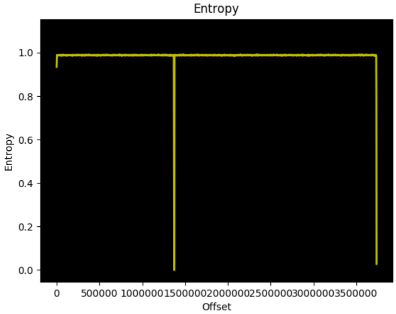
```
user@debian:~/18638-tutorial$ binwalk --term openwrt-18.06.0-ar71xx-generic-wrt1
60nl-squashfs-factory.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: NL16, hardware version:
                              4702, firmware version: 1.0.0, build date:
                              2018-07-30
32            0x20            TRX firmware header, little endian, image size:
                              3735552 bytes, CRC32: 0x7DEDD391, flags: 0x0,
                              version: 1, header size: 28 bytes, loader
                              offset: 0x1C, linux kernel offset: 0x14FFE0,
                              rootfs offset: 0x0
60            0x3C            uImage header, header size: 64 bytes, header CRC:
                              0x8213C649, created: 2018-07-30 16:25:17, image
                              size: 1371501 bytes, Data Address: 0x80060000,
                              Entry Point: 0x80060000, data CRC: 0x3A087825,
                              OS: Linux, CPU: MIPS, image type: OS Kernel
                              Image, compression type: gzip, image name: "MIPS
                              OpenWrt Linux-4.9.111"
124           0x7C            gzip compressed data, maximum compression, from
                              Unix, last modified: 1970-01-01 00:00:00 (null
                              date)
1376256       0x150000        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 2358710 bytes, 1157
                              inodes, blocksize: 262144 bytes, created:
                              2018-07-30 16:25:17

user@debian:~/18638-tutorial$
```

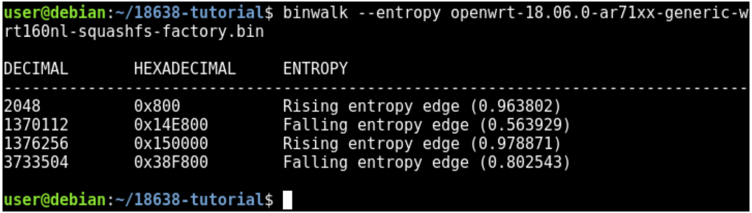
`$ binwalk`

- Regions that contain compressed or encrypted data tend to have high values of **entropy**
- Useful for the inspection of regions that contain data in an unknown format

$ binwalk

- Regions that contain compressed or encrypted data tend to have high values of **entropy**
- Useful for the inspection of regions that contain data in an unknown format





```
user@debian:~/18638-tutorial$ binwalk --entropy openwrt-18.06.0-ar71xx-generic-w
rt160nl-squashfs-factory.bin

DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
2048          0x800           Rising entropy edge (0.963802)
1370112       0x14E800        Falling entropy edge (0.563929)
1376256       0x150000        Rising entropy edge (0.978871)
3733504       0x38F800        Falling entropy edge (0.802543)

user@debian:~/18638-tutorial$ 
```

$ binvis

- We can use `binvis` to generate a **visualization** of the firmware image with space-filling curves in order to identify regions with non-random data

- Coloring scheme:
  - `0x00`: `[0,0,0]`
  - `0xff`: `[255,255,255]`
  - Printable character: `[55,126,184]`
  - Everything else: `[228,26,28]`

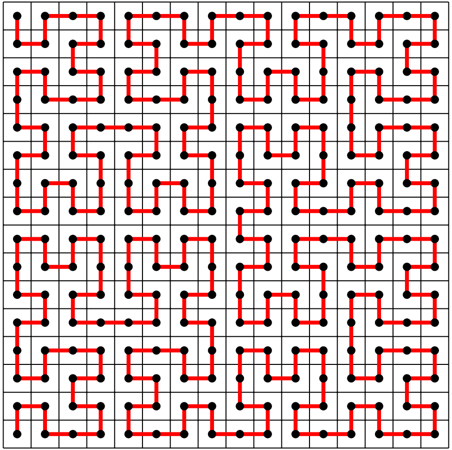
- https://github.com/cortesi/scurve

$ binvis

- We can use `binvis` to generate a **visualization** of the firmware image with space-filling curves in order to identify regions with non-random data

- Coloring scheme:
  - `0x00`: `[0,0,0]`
  - `0xff`: `[255,255,255]`
  - Printable character: `[55,126,184]`
  - Everything else: `[228,26,28]`

- https://github.com/cortesi/scurve

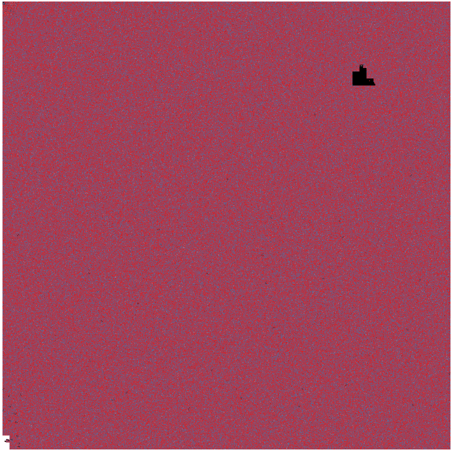


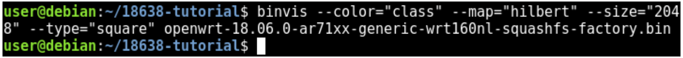
$ binvis

- We can use `binvis` to generate a **visualization** of the firmware image with space-filling curves in order to identify regions with non-random data

- Coloring scheme:
  - `0x00`: `[0,0,0]`
  - `0xff`: `[255,255,255]`
  - Printable character: `[55,126,184]`
  - Everything else: `[228,26,28]`

- https://github.com/cortesi/scurve

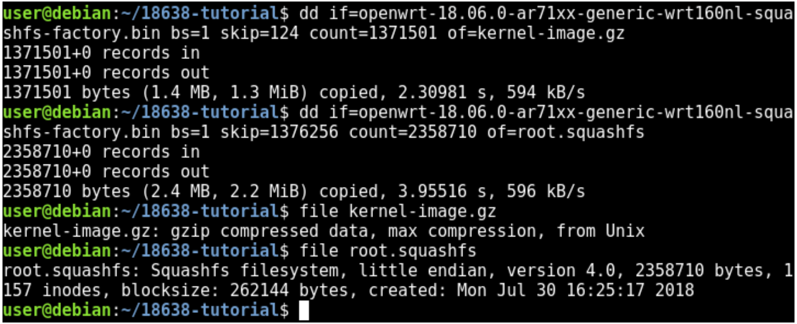




```
user@debian:~/18638-tutorial$ binvis --color="class" --map="hilbert" --size="204
8" --type="square" openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin
user@debian:~/18638-tutorial$ 
```

$ dd

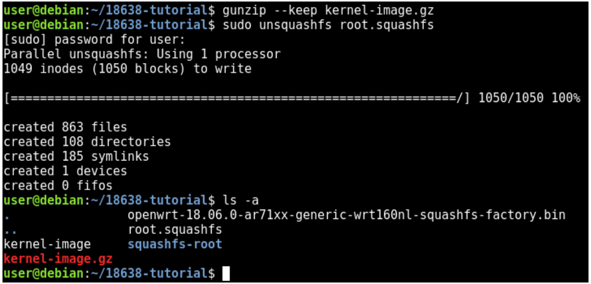
- We can duplicate regions of the firmware image with the dd command:
  - `if` option: Input file
  - `bs` option: Number of bytes in a block (in decimal notation)
  - `skip` option: Number of blocks to skip (in decimal notation)
  - `count` option: Number of blocks to copy (in decimal notation)
  - `of` option: Output file



```
user@debian:~/18638-tutorial$ dd if=openwrt-18.06.0-ar71xx-generic-wrt160nl-squa
shfs-factory.bin bs=1 skip=124 count=1371501 of=kernel-image.gz
1371501+0 records in
1371501+0 records out
1371501 bytes (1.4 MB, 1.3 MiB) copied, 2.30981 s, 594 kB/s
user@debian:~/18638-tutorial$ dd if=openwrt-18.06.0-ar71xx-generic-wrt160nl-squa
shfs-factory.bin bs=1 skip=1376256 count=2358710 of=root.squashfs
2358710+0 records in
2358710+0 records out
2358710 bytes (2.4 MB, 2.2 MiB) copied, 3.95516 s, 596 kB/s
user@debian:~/18638-tutorial$ file kernel-image.gz
kernel-image.gz: gzip compressed data, max compression, from Unix
user@debian:~/18638-tutorial$ file root.squashfs
root.squashfs: Squashfs filesystem, little endian, version 4.0, 2358710 bytes, 1
157 inodes, blocksize: 262144 bytes, created: Mon Jul 30 16:25:17 2018
user@debian:~/18638-tutorial$ █
```

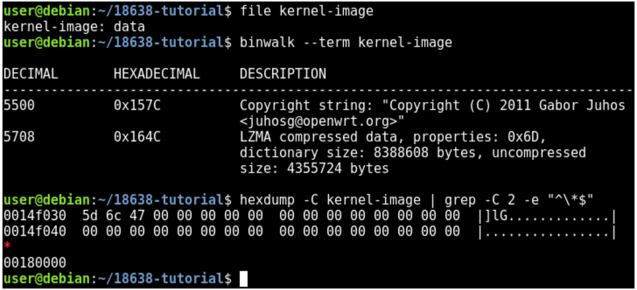
Data extraction tools

- We can extract gzip compressed data with `gunzip` and SquashFS filesystems with `unsquashfs`
- Vendors often use **non-standard** SquashFS filesystems that `unsquashfs` is unable to extract:
  - https://github.com/devttys0/sasquatch
- With the `--extract` option, `binwalk` uses common tools to extract the files that it identified

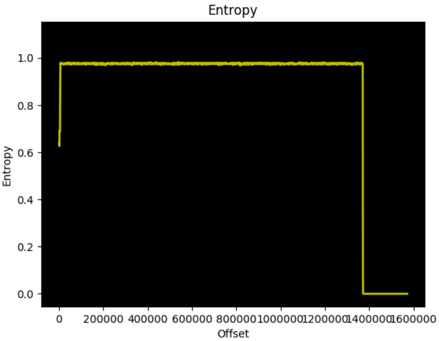


```
user@debian:~/18638-tutorial$ gunzip --keep kernel-image.gz
user@debian:~/18638-tutorial$ sudo unsquashfs root.squashfs
[sudo] password for user:
Parallel unsquashfs: Using 1 processor
1049 inodes (1050 blocks) to write

[=============================================================/] 1050/1050 100%

created 863 files
created 108 directories
created 185 symlinks
created 1 devices
created 0 fifos
user@debian:~/18638-tutorial$ ls -a
.                openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin
..               root.squashfs
kernel-image     squashfs-root
kernel-image.gz
user@debian:~/18638-tutorial$ 
```

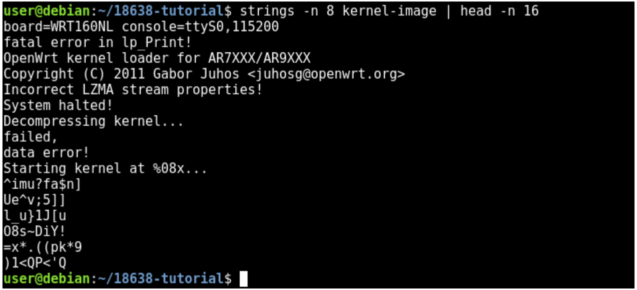
Inspecting the kernel image



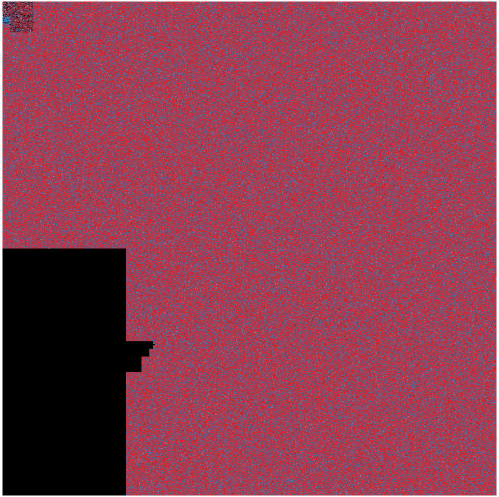
```
user@debian:~/18638-tutorial$ file kernel-image
kernel-image: data
user@debian:~/18638-tutorial$ binwalk --term kernel-image

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
5500          0x157C          Copyright string: "Copyright (C) 2011 Gabor Juhos
                              <juhosg@openwrt.org>"
5708          0x164C          LZMA compressed data, properties: 0x6D,
                              dictionary size: 8388608 bytes, uncompressed
                              size: 4355724 bytes

user@debian:~/18638-tutorial$ hexdump -C kernel-image | grep -C 2 -e "^\*$"
0014f030  5d 6c 47 00 00 00 00 00  00 00 00 00 00 00 00 00  |]lG.............|
0014f040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00180000
user@debian:~/18638-tutorial$ 
```



Inspecting the kernel image

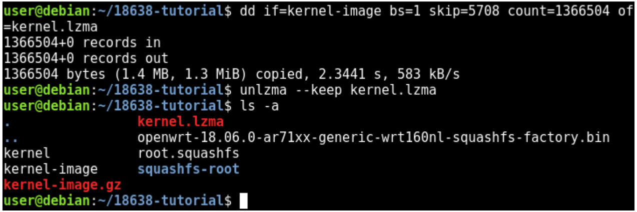


```
user@debian:~/18638-tutorial$ strings -n 8 kernel-image | head -n 16
board=WRT160NL console=ttyS0,115200
fatal error in lp_Print!
OpenWrt kernel loader for AR7XXX/AR9XXX
Copyright (C) 2011 Gabor Juhos <juhosg@openwrt.org>
Incorrect LZMA stream properties!
System halted!
Decompressing kernel...
failed,
data error!
Starting kernel at %08x...
^imu?fa$n]
Ue^v;5]]
l_u}1J[u
08s~DiY!
=x*.((pk*9
)1<QP<'Q
user@debian:~/18638-tutorial$ 
```



Decompressing the kernel

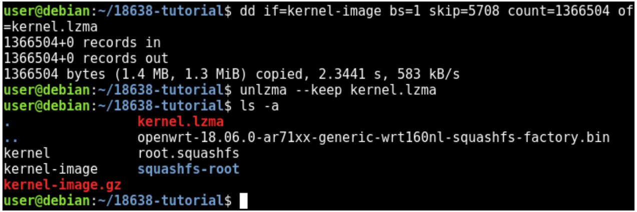
- We can extract LZMA compressed data with the `unlzma` command
- For recursive scanning and extraction of known files, we can use `binwalk` with the `--extract` and `--matryoshka` options, or simply `-eM`



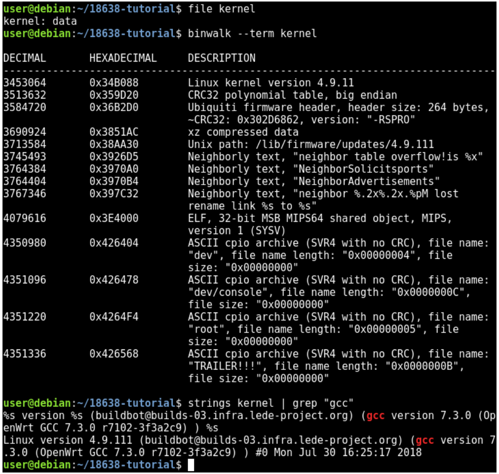
```
user@debian:~/18638-tutorial$ dd if=kernel-image bs=1 skip=5708 count=1366504 of
=kernel.lzma
1366504+0 records in
1366504+0 records out
1366504 bytes (1.4 MB, 1.3 MiB) copied, 2.3441 s, 583 kB/s
user@debian:~/18638-tutorial$ unlzma --keep kernel.lzma
user@debian:~/18638-tutorial$ ls -a
.                 kernel.lzma
..                openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin
kernel            root.squashfs
kernel-image      squashfs-root
kernel-image.gz
user@debian:~/18638-tutorial$ 
```

Decompressing the kernel

- We can extract LZMA compressed data with the `unlzma` command
- For recursive scanning and extraction of known files, we can use `binwalk` with the `--extract` and `--matryoshka` options, or simply `-eM`

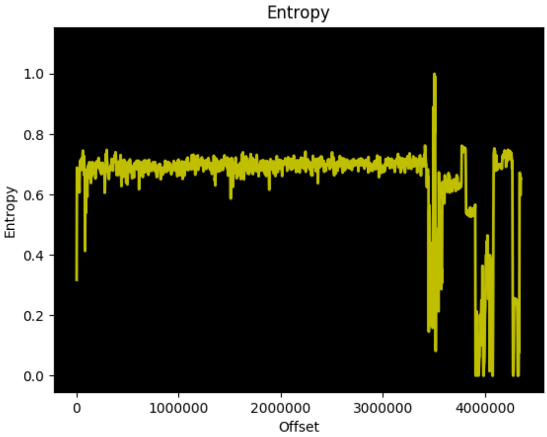


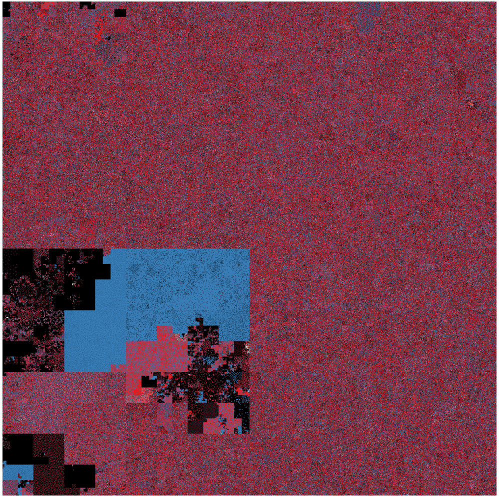
```
user@debian:~/18638-tutorial$ dd if=kernel-image bs=1 skip=5708 count=1366504 of
=kernel.lzma
1366504+0 records in
1366504+0 records out
1366504 bytes (1.4 MB, 1.3 MiB) copied, 2.3441 s, 583 kB/s
user@debian:~/18638-tutorial$ unlzma --keep kernel.lzma
user@debian:~/18638-tutorial$ ls -a
.                 kernel.lzma
..                openwrt-18.06.0-ar71xx-generic-wrt160nl-squashfs-factory.bin
kernel            root.squashfs
kernel-image      squashfs-root
kernel-image.gz
user@debian:~/18638-tutorial$ 
```



```
user@debian:~/18638-tutorial$ file kernel
kernel: data
user@debian:~/18638-tutorial$ binwalk --term kernel

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
3453064       0x34B088        Linux kernel version 4.9.11
3513632       0x359D20        CRC32 polynomial table, big endian
3584720       0x36B2D0        Ubiquiti firmware header, header size: 264 bytes,
                              ~CRC32: 0x302D6862, version: "-RSPRO"
3690924       0x3851AC        xz compressed data
3713584       0x38AA30        Unix path: /lib/firmware/updates/4.9.111
3745493       0x3926D5        Neighborly text, "neighbor table overflow!is %x"
3764384       0x3970A0        Neighborly text, "NeighborSolicitsports"
3764404       0x3970B4        Neighborly text, "NeighborAdvertisements"
3767346       0x397C32        Neighborly text, "neighbor %.2x%.2x.%pM lost
                              rename link %s to %s"
4079616       0x3E4000        ELF, 32-bit MSB MIPS64 shared object, MIPS,
                              version 1 (SYSV)
4350980       0x426404        ASCII cpio archive (SVR4 with no CRC), file name:
                              "dev", file name length: "0x00000004", file
                              size: "0x00000000"
4351096       0x426478        ASCII cpio archive (SVR4 with no CRC), file name:
                              "dev/console", file name length: "0x0000000C",
                              file size: "0x00000000"
4351220       0x4264F4        ASCII cpio archive (SVR4 with no CRC), file name:
                              "root", file name length: "0x00000005", file
                              size: "0x00000000"
4351336       0x426568        ASCII cpio archive (SVR4 with no CRC), file name:
                              "TRAILER!!!", file name length: "0x0000000B",
                              file size: "0x00000000"

user@debian:~/18638-tutorial$ strings kernel | grep "gcc"
%s version %s (buildbot@builds-03.infra.lede-project.org) (gcc version 7.3.0 (Op
enWrt GCC 7.3.0 r7102-3f3a2c9) ) %s
Linux version 4.9.111 (buildbot@builds-03.infra.lede-project.org) (gcc version 7
.3.0 (OpenWrt GCC 7.3.0 r7102-3f3a2c9) ) #0 Mon Jul 30 16:25:17 2018
user@debian:~/18638-tutorial$ 
```

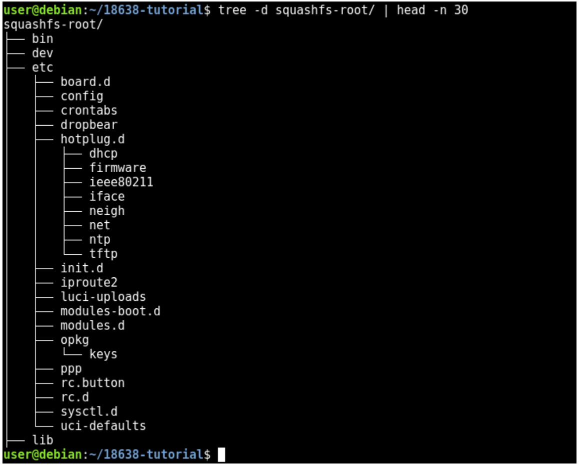
Inspecting the kernel





Inspecting the filesystem

- What to look for in the filesystem?
  - Password files
  - Encryption keys
  - Public key certificates
  - Executable files
  - Configuration files
  - Interesting keywords
- We can use `firmwalker` to search for some common files and keywords in the filesystem:
  - https://github.com/craigz28/firmwalker



```
user@debian:~/18638-tutorial$ tree -d squashfs-root/ | head -n 30
squashfs-root/
├── bin
├── dev
├── etc
│   ├── board.d
│   ├── config
│   ├── crontabs
│   ├── dropbear
│   ├── hotplug.d
│   │   ├── dhcp
│   │   ├── firmware
│   │   ├── ieee80211
│   │   ├── iface
│   │   ├── neigh
│   │   ├── net
│   │   ├── ntp
│   │   └── tftp
│   ├── init.d
│   ├── iproute2
│   ├── luci-uploads
│   ├── modules-boot.d
│   ├── modules.d
│   ├── opkg
│   │   └── keys
│   ├── ppp
│   ├── rc.button
│   ├── rc.d
│   ├── sysctl.d
│   └── uci-defaults
├── lib
user@debian:~/18638-tutorial$ 
```

Password files

- Usually, the system's accounts can be found in the `/etc/passwd` file and their hashed passwords are stored in the `/etc/shadow` file

- For more information regarding the format of those files:
  - `$ man 5 passwd`
  - `$ man 5 shadow`
  - `$ man 3 crypt`

- Traditional DES-based password hashes can be easily cracked with `john`:
  - http://www.openwall.com/john/

Encryption keys

- Many devices contain hard-coded private keys in their firmware in order to support HTTPS:
  - http://www.devttys0.com/2010/12/breaking-ssl-on-embedded-devices/

Encryption keys

- Many devices contain hard-coded private keys in their firmware in order to support HTTPS:
  - http://www.devttys0.com/2010/12/breaking-ssl-on-embedded-devices/

- Multiple devices may be using the same encryption keys, sometimes even devices of different vendors:
  - https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin
  - https://www.sec-consult.com/en/blog/2016/09/house-of-keys-9-months-later-40-worse/index.html

Encryption keys

- Many devices contain hard-coded private keys in their firmware in order to support HTTPS:
  - http://www.devttys0.com/2010/12/breaking-ssl-on-embedded-devices/
- Multiple devices may be using the same encryption keys, sometimes even devices of different vendors:
  - https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin
  - https://www.sec-consult.com/en/blog/2016/09/house-of-keys-9-months-later-40-worse/index.html
- Datasets of private keys that were found in embedded systems:
  - https://github.com/devttys0/littleblackbox
  - https://github.com/sec-consult/houseofkeys

Public key certificates

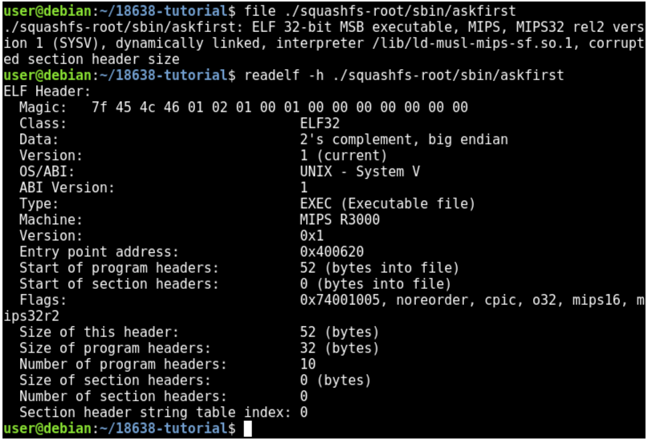
- We can process private keys, public keys, and X.509 certificates with the `openssl` program
- For example, we can view the contents of an X.509 certificate in PEM format with the following command:
  - `$ openssl x509 -in certificate.pem -text -noout`

Public key certificates

- We can process private keys, public keys, and X.509 certificates with the `openssl` program
- For example, we can view the contents of an X.509 certificate in PEM format with the following command:
  - `$ openssl x509 -in certificate.pem -text -noout`
- We can estimate the number of Internet-connected devices that use the same public key certificate by searching for its fingerprint on computer search engines:
  - https://www.shodan.io/
  - https://censys.io/

Executable files

- We can examine executable files in ELF format with the `readelf` command
- For example, with the `-h` option, `readelf` displays the information that is contained in the header of the ELF file
- We can disassemble ELF files with tools like `radare2`:
  - https://github.com/radare/radare2

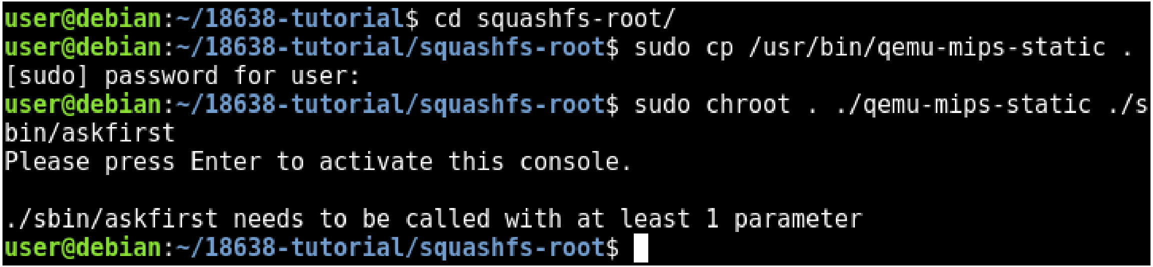


```
user@debian:~/18638-tutorial$ file ./squashfs-root/sbin/askfirst
./squashfs-root/sbin/askfirst: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 vers
ion 1 (SYSV), dynamically linked, interpreter /lib/ld-musl-mips-sf.so.1, corrupt
ed section header size
user@debian:~/18638-tutorial$ readelf -h ./squashfs-root/sbin/askfirst
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 01 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:                       1
  Type:                              EXEC (Executable file)
  Machine:                           MIPS R3000
  Version:                           0x1
  Entry point address:               0x400620
  Start of program headers:          52 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x74001005, noreorder, cpic, o32, mips16, m
ips32r2
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         10
  Size of section headers:           0 (bytes)
  Number of section headers:         0
  Section header string table index: 0
user@debian:~/18638-tutorial$ 
```

QEMU user mode emulation

- We can use QEMU in user mode to execute binary files that were compiled for a different computer architecture than that of our host system:
    - https://www.qemu.org/

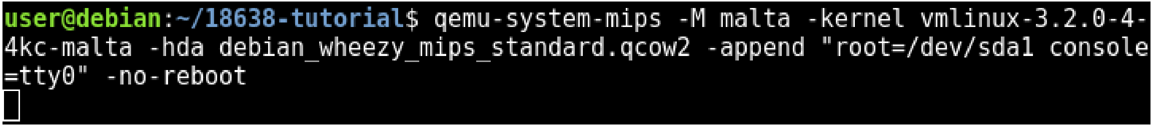
- We use the `chroot` command to execute the ELF file with the extracted SquashFS filesystem as root directory



```
user@debian:~/18638-tutorial$ cd squashfs-root/
user@debian:~/18638-tutorial/squashfs-root$ sudo cp /usr/bin/qemu-mips-static .
[sudo] password for user:
user@debian:~/18638-tutorial/squashfs-root$ sudo chroot . ./qemu-mips-static ./s
bin/askfirst
Please press Enter to activate this console.

./sbin/askfirst needs to be called with at least 1 parameter
user@debian:~/18638-tutorial/squashfs-root$ 
```

QEMU full system emulation

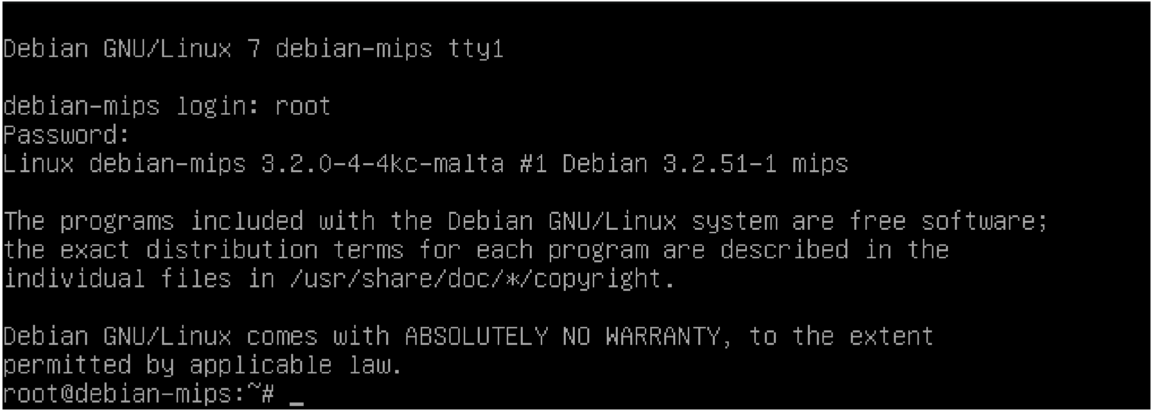
- QEMU also supports full system emulation using prebuilt images:
  - https://people.debian.org/~aurel32/qemu/



```
user@debian:~/18638-tutorial$ qemu-system-mips -M malta -kernel vmlinux-3.2.0-4-
4kc-malta -hda debian_wheezy_mips_standard.qcow2 -append "root=/dev/sda1 console
=tty0" -no-reboot
```

QEMU full system emulation

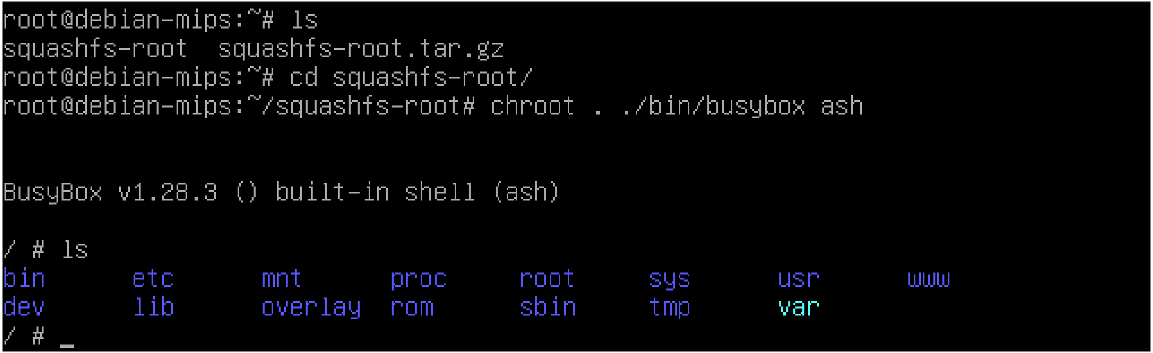
- QEMU also supports full system emulation using prebuilt images:
  - https://people.debian.org/~aurel32/qemu/



```
Debian GNU/Linux 7 debian-mips tty1

debian-mips login: root
Password:
Linux debian-mips 3.2.0-4-4kc-malta #1 Debian 3.2.51-1 mips

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@debian-mips:~# _
```

QEMU full system emulation

- We can copy the extracted filesystem in the hard disk image and then initiate a command interpreter (shell) with `chroot`



```
root@debian-mips:~# ls
squashfs-root  squashfs-root.tar.gz
root@debian-mips:~# cd squashfs-root/
root@debian-mips:~/squashfs-root# chroot . ./bin/busybox ash


BusyBox v1.28.3 () built-in shell (ash)

/ # ls
bin      etc      mnt      proc     root     sys      usr      www
dev      lib      overlay  rom      sbin     tmp      var
/ # _
```

General security concerns

- Is there any information leakage from the device?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?
- Is the device susceptible to replay attacks?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?
- Is the device susceptible to replay attacks?
- Is the firmware image digitally signed?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?
- Is the device susceptible to replay attacks?
- Is the firmware image digitally signed?
- Is the device running any unnecessary services?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?
- Is the device susceptible to replay attacks?
- Is the firmware image digitally signed?
- Is the device running any unnecessary services?
- Are there any backdoors in the firmware?

General security concerns

- Is there any information leakage from the device?
- Does the device accept unauthenticated commands?
- Is the device susceptible to replay attacks?
- Is the firmware image digitally signed?
- Is the device running any unnecessary services?
- Are there any backdoors in the firmware?
- Is the device using outdated software with known vulnerabilities?