Bangladesh University of Engineering and Technology

CSE 406 - Computer Security

Report On

BINWALK

Ruhul Azgor
1805091

Md. Nazmul Islam Ananto
1805093

Supervisor: Ruhan Islam

Contents

1 Introduction

Binwalk is a tool for searching a given binary image for embedded files and executable code. Specifically, it is designed for identifying files and code embedded inside of firmware images. Binwalk uses the libmagic library, so it is compatible with magic signatures created for the Unix file utility.

1.1 Usage

- **Signature Finding:** Identifying unique patterns or signatures within a binary file, often used in digital forensics to recognize file formats or known malware.
- **Entropy Analysis:** Measuring the randomness or disorder within data; it's used in cybersecurity to assess the unpredictability of encryption keys or to detect compressed or encrypted content.
- **Forensic Analysis:** Examining digital evidence (e.g., computer files) to gather information for legal purposes, such as in criminal investigations.
- **Reverse Engineering:** Reverse engineering is the process of dissecting and understanding the inner workings of a product, software, or system by analyzing its structure, behavior, and functionality. This method involves examining the compiled code, binaries, or hardware components to unveil their logic, algorithms, and design. Often used to comprehend proprietary or closed-source systems, reverse engineering aids in uncovering hidden vulnerabilities, improving interoperability, and developing compatible solutions.
- **Developing Compatible Software for Closed-Source Systems:** Creating software that can interact with or run on systems whose source code is not publicly available or accessible. Or even when there is no good documentation.
- **Discovering Hidden Vulnerabilities:** Finding and exposing weaknesses or security flaws in software or systems, which can then be fixed to improve security.

2 Overview Of Source Code

Binwalk started a while ago. Many people from the computer security community have contributed to make it better and the most prominent contributor is its creator, Craig HeffnerS.

2.1 Language

Binwalk was originally written in Python 2.7 but has since transitioned to Python 3 in its latest release, reflecting the ongoing development and adaptation of the tool to modern programming standards

2.2 Repository

Binwalk's source code is available on GitHub.
It can be accessed at: `https://github.com/ReFirmLabs/binwalk`
Repository Owner: ReFirm Labs

2.3 Key Components

1. **Core Modules**: These modules provide the foundational functionalities of Binwalk, including signature scanning, extraction, and data analysis.
2. **Signature Definitions**: Binwalk relies on a collection of signature definitions to identify specific file formats within binary data. These signature definitions are stored in the `src/signature` directory and are written in Python.

3. **Extraction Modules**: These modules handle the extraction of data from binary files based on identified signatures.
4. **Compression and Encryption**: Binwalk supports decompression and decryption of various formats, and the source code contains modules to handle these processes.
5. **Visualization**: Binwalk provides visualization capabilities to better understand the layout and structure of binary files.
6. **User Interface**: The source code includes command-line interface (CLI) components to interact with Binwalk and execute its various features.

2.4 Directories and Major Files

binwalk/ The main directory that contains the Binwalk source code and related files.

deps/ Dependencies used by Binwalk, such as the Capstone disassembly framework.

docs/ Documentation and guides related to Binwalk.

src/ The core source code of Binwalk is located here.

binwalk.py The main entry point of Binwalk. Contains the CLI definition, argument parsing, and core logic.

core/ Directory containing the core modules and functionalities of Binwalk.

modules/ Various modules that implement specific features like signature scanning, extraction, and more.

signatures/ Contains pre-defined signature definitions for various file types.

plugins/ Additional plugins that extend Binwalk's capabilities.

docopt/ The Docopt library for parsing command-line arguments.

data/ Various data files used by Binwalk, including magic signatures.

scripts/ Useful scripts and utilities.

common.py Common functions and definitions used across the codebase.

version.py Version information for Binwalk.

2.5 Functionality Overview

Signature Scanning Implemented in `modules/signature.py`, this core functionality scans for file signatures, opcodes, and raw data within binary files.

Extraction `modules/extractor.py` handles data extraction based on identified signatures, supporting various methods and utilities.

Entropy Analysis Entropy calculation of data sections is implemented in `modules/entropy.py`.

Visualization Visualization capabilities, including graph and plot generation, are provided by `modules/graph.py`.

Custom Signatures Users can define custom signatures using `--magic` and `--raw` options, managed by `modules/custom.py`.

Recursive Scanning Managed by `--matryoshka`, enabling recursive scanning and extraction from embedded files.

Firmware Analysis Firmware analysis functionalities span `modules/firmware.py`, `modules/extractor.py`, and more.

2.6 Integration of External Tools

Binwalk integrates external tools like `unsquashfs`, `mtd-utils`, `tar`, `gzip`, and others for specific extraction tasks.

2.7 Plugin System

Binwalk features a plugin system in the `plugins/` directory, allowing additional functionalities to be added to the tool.

3 Usage Options

If we run 'binwalk -h' it gives us the usages.

3.1 Signature Scan Options

-B, –signature Scan target file(s) for common file signatures

-R, –raw=<str> Scan target file(s) for the specified sequence of bytes

-A, –opcodes Scan target file(s) for common executable opcode signatures

-m, –magic= <file> Specify a custom magic file to use

-b, –dumb Disable smart signature keywords

-I, –invalid Show results marked as invalid

-x, –exclude=<str> Exclude results that match <str>

-y, –include=<str> Only show results that match <str>

3.2 Extraction Options

-e, –extract Automatically extract known file types

-D, –dd=<type:ext:cmd> Extract <type>signatures, give the files an extension of <ext>, and execute <cmd>

-M, –matryoshka Recursively scan extracted files

-d, –depth=<int> Limit matryoshka recursion depth (default: 8 levels deep)

-C, –directory=<str> Extract files/folders to a custom directory (default: current working directory)

-j, –size=<int> Limit the size of each extracted file

-n, –count=<int> Limit the number of extracted files

-r, –rm Delete carved files after extraction

-z, –carve Carve data from files, but don't execute extraction utilities

-V, –subdirs Extract into sub-directories named by the offset

3.3 Entropy Options

-E, –entropy Calculate file entropy

-F, –fast Use faster, but less detailed, entropy analysis

-J, –save Save plot as a PNG

-Q, –nlegend Omit the legend from the entropy plot graph

-N, –nplot Do not generate an entropy plot graph

-H, –high=¡float¿ Set the rising edge entropy trigger threshold (default: 0.95)

-L, –low=¡float¿ Set the falling edge entropy trigger threshold (default: 0.85)

3.4 Binary Diffing Options

-W, –hexdump Perform a hexdump / diff of a file or files

-G, –green Only show lines containing bytes that are the same among all files

-i, –red Only show lines containing bytes that are different among all files

-U, –blue Only show lines containing bytes that are different among some files

-u, –similar Only display lines that are the same between all files

-w, –terse Diff all files, but only display a hex dump of the first file

3.5 Raw Compression Options

-X, –deflate Scan for raw deflate compression streams

-Z, –lzma Scan for raw LZMA compression streams

-P, –partial Perform a superficial, but faster, scan

-S, –stop Stop after the first result

3.6 General Options

-l, –length=<int> Number of bytes to scan

-o, –offset=<int> Start scan at this file offset

-O, –base=<int> Add a base address to all printed offsets

-K, –block=<int> Set file block size

-g, –swap=<int> Reverse every n bytes before scanning

-f, –log=<file> Log results to file

-c, –csv Log results to file in CSV format

-t, –term Format output to fit the terminal window

-q, –quiet Suppress output to stdout

-v, –verbose Enable verbose output

-h, –help Show help output

-a, –finclude=<str> Only scan files whose names match this regex

-p, –fexclude=<str> Do not scan files whose names match this regex

-s, –status=<int> Enable the status server on the specified port

4 Demonstration

4.1 Installation

It is pre-installed on the Kali Linux operating system. Just remember Binwalk's older version is not compatible with the latest versions, hence it is suggested to uninstall the older version before installing the latest version to avoid any API conflict.

If you want to install it on a Linux system, you need to install a python3 interpreter as a prerequisite.

```
sudo apt-get update
sudo apt-get install python3
```

Then download the Binwalk binary from the download link , Navigate to unzip the download directory, and use the below command for installation:

```
sudo python3 setup.py install
```

4.2 Help

Command: `binwalk -h`

Description

This command displays all options

Usage

```
binwalk -h
```

Example

```
seed@ruhu:~/Binwalk/Tplink$ binwalk -h

Binwalk v2.3.3
Craig Heffner, ReFirmLabs
https://github.com/ReFirmLabs/binwalk

Usage: binwalk [OPTIONS] [FILE1] [FILE2] [FILE3] ...

Signature Scan Options:
    -B, --signature             Scan target file(s) for common file signatures
    -R, --raw=<str>             Scan target file(s) for the specified sequence of bytes
    -A, --opcodes               Scan target file(s) for common executable opcode signatures
    -m, --magic=<file>          Specify a custom magic file to use
    -b, --dumb                  Disable smart signature keywords
    -I, --invalid               Show results marked as invalid
    -x, --exclude=<str>         Exclude results that match <str>
    -y, --include=<str>         Only show results that match <str>
```

Figure 1: Using the `flag -h` to get all the options.

4.3 Signature Finding

Command: `binwalk` <firmware > **or binwalk -B** <firmware >

Description

Scan to identify code, files, and other information

Usage

`binwalk -B <firmware>`

Example

Figure 2: Scan to identify code, files, and other information

Explanation

Here cool_cat.png has a image file with a executable appended in the last. Binwalk finds the executable file's position along with the image file and outputs the information.

4.4 All signatures [inc. invalid ones]

Command: `binwalk -I <firmware >`

Description

Useful when binwalk is treating a valid file as invalid, may mislead with garbages

Usage

`binwalk -I <firmware>`

Example

Explanation

Here cool_cat.png has a image file with a executable appended in the last. But binwalk finds a lot of other things.Useful when binwalk is treating a valid file as invalid.

4.5 Raw String Finding

Command: `binwalk -R '<string or escaped in octal>' <filename>`

Description

This allows to search the specified file(s) for a custom string.

Usage

`binwalk -R '<string or escaped in octal>' <filename>`

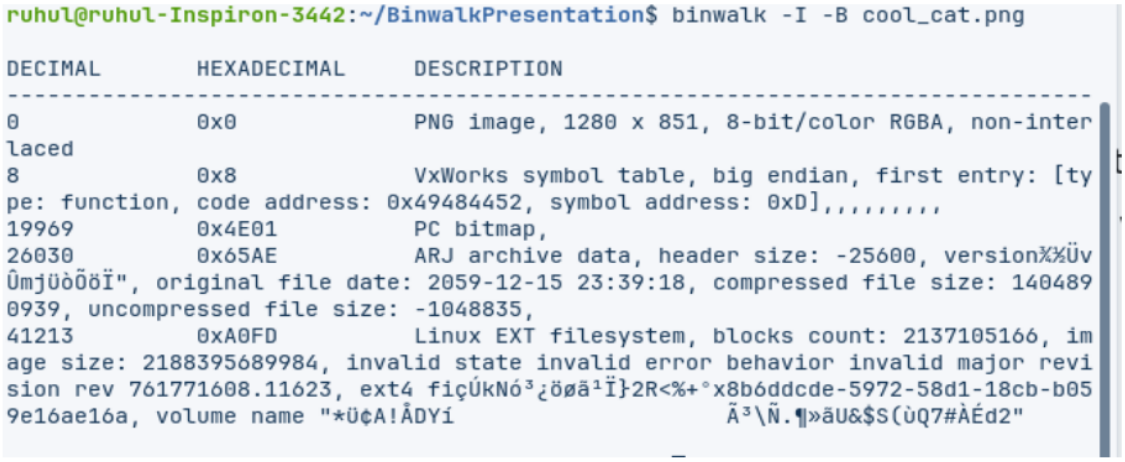


Figure 3: Scan to all signature

Example

Figure 4: Scan to raw string match

Explanation

Its find the string 'hello' in the cool_cat.png.

4.6 Opcode Analysis

Command: `binwalk -A <file_name>or binwalk -- opcode <file_name>`

Description

Searches for opcode to determine the architecture of the file. Can be misleading.

Usage

`binwalk -A <file_name> or binwalk -- opcode <file_name>`

Example

Explanation

It finds Intel x86 instructions which is in the cool_cat.png. Which is true indeed.

```
ruhul@ruhul-Inspiron-3442:~/BinwalkPresentation$ binwalk -A cool_cat.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
446505        0x6D029         Intel x86 instructions, function prologue
```

Figure 5: Searches for opcode

4.7 Extraction

Command: `binwalk -e <firmware >`

Command: `binwalk -e --dd=".*" firmware.bin`

-dd =<type[:ext[:cmd]]>extracts files identified during a –signature scan

Description

Extract files from firmware. Loads common –dd extraction rules from a predefined file

Usage

`binwalk -e <firmware>`

Example

```
(base) azgor@azgor-MS-7B98:~/BinwalkPresentation$ binwalk -e hello.zip

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, uncompressed size:
 15588, name: hello
2903          0xB57           End of Zip archive, footer length: 22
```

```
(base) azgor@azgor-MS-7B98:~/BinwalkPresentation$ cd _hello.zip.extracted/
(base) azgor@azgor-MS-7B98:~/BinwalkPresentation/_hello.zip.extracted$ ls
0.zip  hello
```

Figure 6: Extract files from firmware

Explanation

It extract the 'hello.zip' which constains a 'hello' executable file.

4.8 Recursive Extraction

Command: `binwalk -Me <firmware-image>`

Description

Extract files from firmware recursively

Usage

`binwalk -Me <firmware-image>`

Example

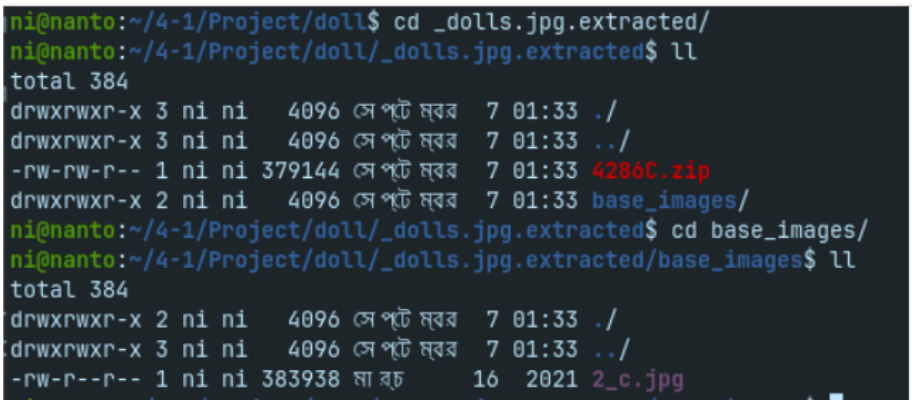


Figure 7: First iteration: contains another zip

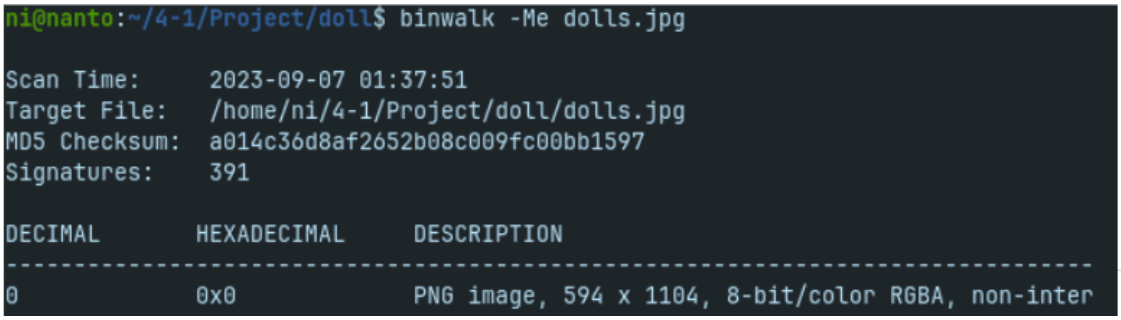


Figure 8: Extraction recursively

Explanation

It doesn't extract the inner zip without recursive option. When recursive option is given it extracts recursively.

4.9 Entropy

Command: `binwalk -E`

Description

Performs an entropy analysis on the input file(s), prints raw entropy data and generates entropy graphs

Usage

```
binwalk -E <firmware-image>
```

Output

```
DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
0             0x0             Falling entropy edge (0.704053)
3280896       0x321000        Falling entropy edge (0.833766)
```

```
ni@nanto:~/4-1/Project/doll$ cd _dolls.jpg-0.extracted/base_images/_2_c.jpg.extr
acted/base_images/_3_c.jpg.extracted/base_images/_4_c.jpg.extracted/
ni@nanto:~/4-1/Project/doll/_dolls.jpg-0.extracted/base_images/_2_c.jpg.extracte
d/base_images/_3_c.jpg.extracted/base_images/_4_c.jpg.extracted$ ll
total 16
drwxrwxr-x 2 ni ni 4096 সেপ্টে ম্বর  7 01:37 ./
drwxrwxr-x 3 ni ni 4096 সেপ্টে ম্বর  7 01:37 ../
-rw-rw-r-- 1 ni ni  230 সেপ্টে ম্বর  7 01:37 136DA.zip
-rw-r--r-- 1 ni ni   81 মা র্চ     16  2021 flag.txt
ni@nanto:~/4-1/Project/doll/_dolls.jpg-0.extracted/base_images/_2_c.jpg.extracte
d/base_images/_3_c.jpg.extracted/base_images/_4_c.jpg.extracted$ cat flag.txt
picoCTF{336cf6d51c9d9774fd37196c1d7320ff}ni@nanto:~/4-1/Project/doll/_dolls.jpg-
```

Figure 9: Extraction recursively

```
3438592       0x347800        Falling entropy edge (0.766975)
3573760       0x368800        Falling entropy edge (0.839854)
3663872       0x37E800        Falling entropy edge (0.848954)
3708928       0x389800        Falling entropy edge (0.848504)
3753984       0x394800        Falling entropy edge (0.131163)
3866624       0x3B0000        Falling entropy edge (0.841466)
3956736       0x3C6000        Falling entropy edge (0.846655)
4046848       0x3DC000        Falling entropy edge (0.739854)
...
```

Example

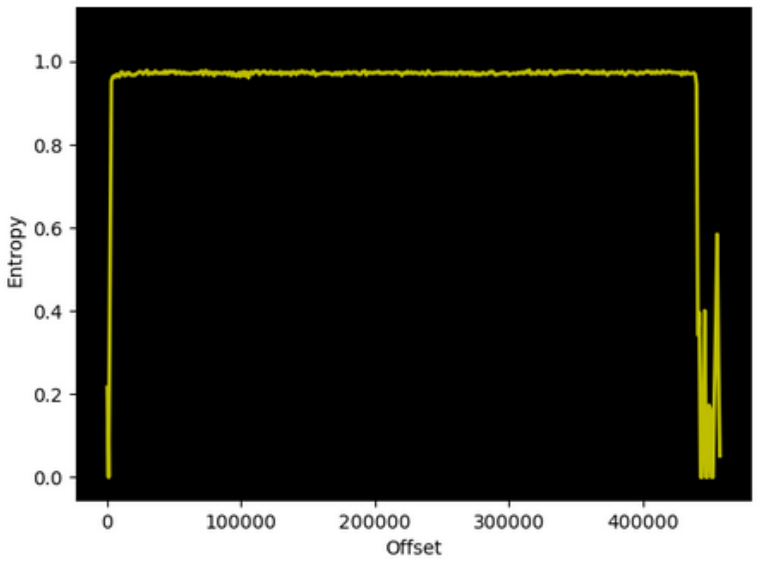


Figure 10: Entropy Analysis

Explanation

High value of entropy indicates the file is zipped. That means more randomness.Usually $entropy > .7$ indicates zip file. This file contains both zip file at the start and regular file in the end.

Entropy of regular file

Figure 4.9

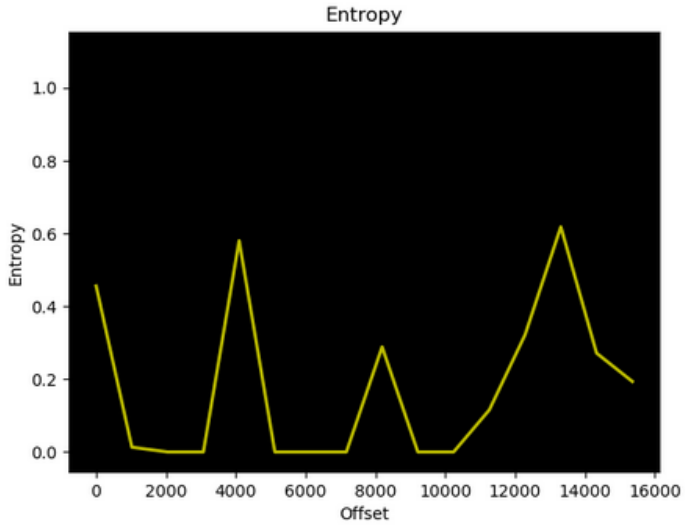


Figure 11: Entropy of regular file

Entropy of zipped file

Figure 4.9

4.10 Finding Custom magic signature

Command: `binwalk -m <file.mgc><firmware.bin>`

Description

Search for customize file signature.

Magic Signature

To understand the basic format of a signature, let's create a new signature for a fictitious firmware header. The header structure is:

```
struct header
{
   char magic[4];        //Magic bytes are: 'SIG0'
   char description[12];
   int32_t header_size;
   int32_t image_size;
   int32_t creation_date;
```

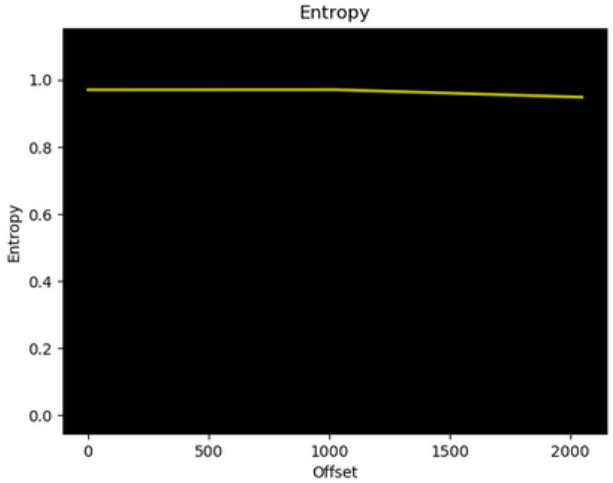


Figure 12: Entropy of zipped file

```
};
```

The resulting magic signature for this header format looks like:

```
0    string    SIG0     SIG0 firmware header,
>4   string    x        description: "%s",
>16  lelong    x        header size: %d,
>20  lelong    x        size: %d,
>24  ledate    x        date: %s
```

More info Magic Signature

Usage

```
binwalk -m <file.mgc> <firmware.bin
```

Example

```
ruhul@ruhul-Inspiron-3442:~/BinwalkPresentation$ binwalk -m foobar.mgc magicFile
.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             SIG0 firmware header, description: "This is a samp
le firmware header", header size: 1701605485, size: 1919510048, date: 2030-10-23
 12:49:49
```

Figure 13: Scan for customized signature

Explanation

The 'magicFile' contains the header 'SIGO'

4.11 Signatures that match the specified include filter

Command: `binwalk -y 'signarure' firmware.bin`

Description

Useful when searching only for specific signatures or types of signature

Usage

```
binwalk -y 'filesystem' firmware.bin # only search for filesystem signatures
```

Example

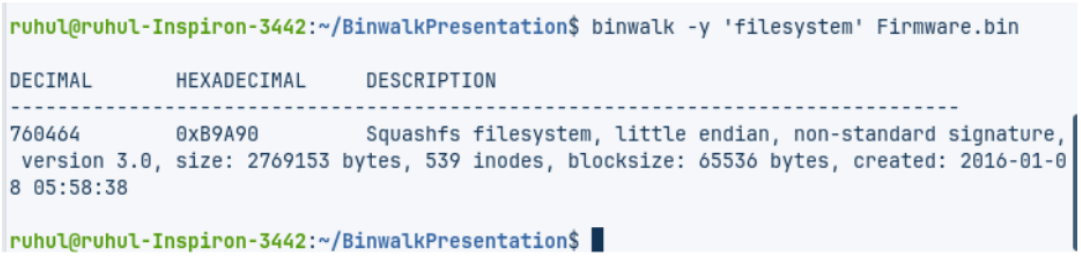


Figure 14: Only search for filesystem signatures

Explanation

In this example, we only scans the filesystem.

4.12 Excludes signatures that match the specified exclude filter

Command: `binwalk -x 'signarure' firmware.bin`

Description

Useful for excluding unneeded or uninteresting results

Usage

```
binwalk -x 'mach-o' -x '^hp' firmware.bin # exclude HP calculator and OSX mach-o signatures
```

Example

Explanation

In this example, Binwalk excludes the SIGO signatures.

4.13 Hexdump (Compare Bytes

Command: `binwalk -W`

Description

Performs a hex dump (Comparison) of the input file(s) and color-codes bytes. Red bytes indicate mismatch and green match

Figure 15: Exclude the SIGO signature

Usage

`Command:binwalk -W --block=8 --length=64 firmware1.bin firmware2.bin firmware3.bin`

Example

Figure 16: Generate differences between firmware images

Flags

-G = Only display lines that contain green bytes
-i = Only display lines that contain red bytes

4.14 Upgrade

Command: `sudo binwalk -u`

Description

Upgrade to the latest version

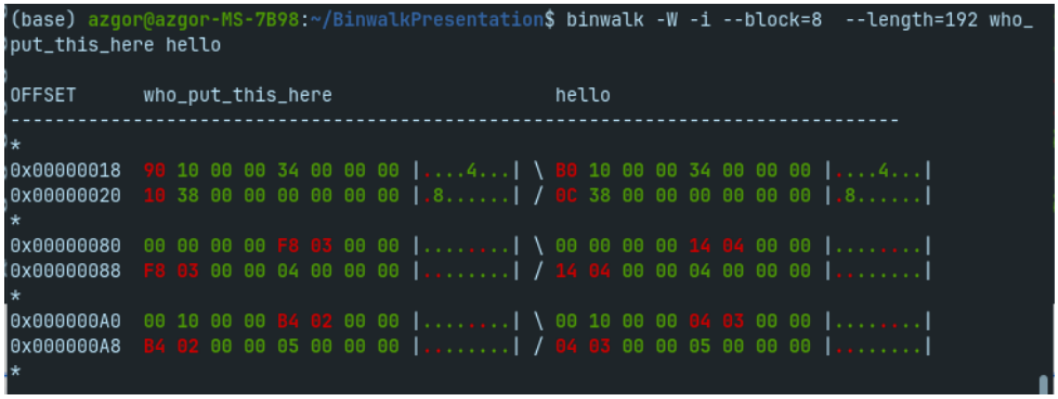


Figure 17: Only red lines

Usage

```
sudo binwalk -u
```

4.15 Verbose

Command: `binwalk -verbose`

Description

Verbose Output

Usage

```
binwalk --verbose <firmware-image>
```

Example

Figure 18: Verbose Output

4.16 Log

Command: `binwalk -f file.log`

Description

Capture log files

Usage

`binwalk -f file.log <firmware-image>`

Example

```
seed@ruhu:~/Binwalk/Tplink$ ls
'Archer AXE5400(USW)_V1_221110'  'How to upgrade TP-Link Wireless Router.pdf'   _firmware.bin.extracted   firmware.bin
'GPL License Terms.pdf'           _firmware.bin-0.extracted                     file.log                  firmware2.bin
seed@ruhu:~/Binwalk/Tplink$ cat file.log

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
4697          0x1259          Flattened device tree, size: 1208 bytes, version: 17
29769         0x7449          SHA256 hash constants, little endian
64665         0xFC99          CRC32 polynomial table, little endian
78009         0x130B9         Flattened device tree, size: 1748 bytes, version: 17
341065        0x53449         SHA256 hash constants, little endian
375961        0x5BC99         CRC32 polynomial table, little endian
389305        0x5F0B9         Flattened device tree, size: 1748 bytes, version: 17
591577        0x906D9         SHA256 hash constants, little endian
652297        0x9F409         CRC32 polynomial table, little endian
```

Figure 19: Capture log files

4.17 Formated Output

Command: `binwalk -t <firmware-image>`

Description

Format output to a current terminal

Usage

`binwalk -t <firmware-image>`

Output

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
4697          0x1259          device tree image (dtb)
29769         0x7449          SHA256 hash constants, little endian
64665         0xFC99          CRC32 polynomial table, little endian
78009         0x130B9         device tree image (dtb)
...
12977493      0xC60555        device tree image (dtb)
13035569      0xC6E831        device tree image (dtb)
13093445      0xC7CA45        device tree image (dtb)
13151921      0xC8AEB1        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 30904328 bytes, 4042
                              inodes, blocksize: 131072 bytes, created:
                              2022-12-08 12:02:10
```

4.18 Disassembly

Command: `binwalk --disasm <firmware-image>`

Description

To display the CPU architecture of binary

Usage

```
binwalk --disasm <firmware-image>
```

Output

```
General Error: Cannot open file --disasm
(CWD: /home/seed/CSE_406/Project/TP-Link-Firmware)
: [Errno 2] No such file or directory: '--disasm'
```

This function is not working for some reason.

4.19 Endianness

Command: `binwalk -y "endian" <firmware-image>`

Description

To display the Endianness of binary

Usage

```
binwalk -y "endian"  <firmware-image>
```

The output looks like this -

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
29769         0x7449          SHA256 hash constants, little endian
64665         0xFC99          CRC32 polynomial table, little endian
341065        0x53449         SHA256 hash constants, little endian
375961        0x5BC99         CRC32 polynomial table, little endian
...
9761584       0x94F330        CRC32 polynomial table, little endian
12121884      0xB8F71C        SHA256 hash constants, little endian
13151921      0xC8AEB1        Squashfs filesystem, little endian,
version 4.0, compression:xz, size: 30904328 bytes, 4042 inodes,
blocksize: 131072 bytes, created: 2022-12-08 12:02:10
```

4.20 Recursively Decompress

Command: `binwalk -reM <firmware-image>`

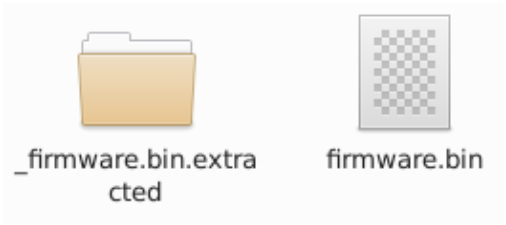
Description

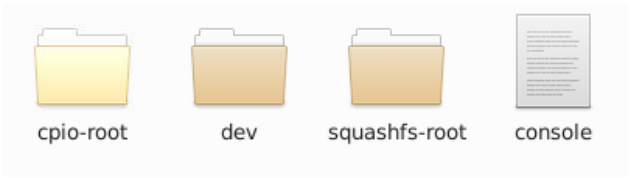
Extract firmware recursively and decompress the file.

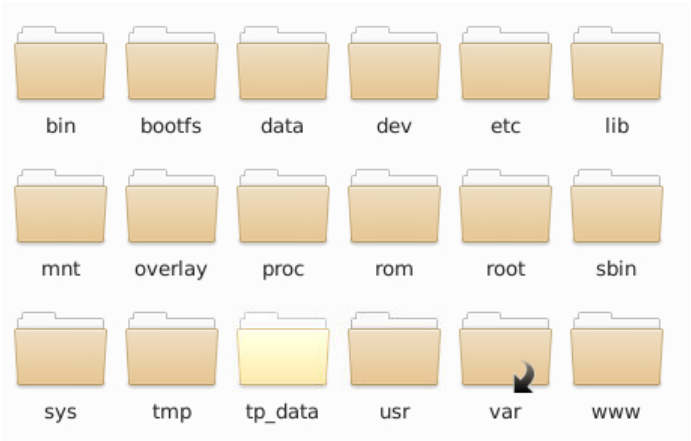
Usage

```
binwalk -reM <firmware-image>
```

This command creates extracted folder and recursively extracts and fills out the folders inside. The file structure looks like this - Inside the squashfs-root folder, there is a whole firmware system.







5 Demo with CTF Problems

5.1 Matryoshka Doll

We will be solving this problem from picoCTF.

Matryoshka dolls are a set of wooden dolls of decreasing size placed one inside another. What's the final one? The image looks like this -

As we can see it looks like a russian doll where there is usually another doll inside and so on. So we might want to recursively extract the dolls.png file.
So we run -

```
binwalk -e -M dolls.jpg
```

The output looks something like this -

Figure 20: CTF Demo - dolls

```
Scan Time:     2023-08-18 18:39:54
Target File:   /home/seed/CSE_406/Project/Matryoshka_doll/dolls.jpg
MD5 Checksum:  a014c36d8af2652b08c009fc00bb1597
Signatures:    391

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 594 x 1104, 8-bit/color RGBA, non-interlaced
3226          0xC9A           TIFF image data, big-endian, offset of first image directory: 8
...
```

Now if we go deeper and deeper into the extracted folders, we'd get a file called 'flag.txt' which would contain our desired flag. File path -

```
_dolls.jpg.extracted/base_images/_2_c.jpg.extracted/base_images/_3_c.jpg.extracted/base_images/_4_c.jp
```

And the flag is -

```
picoCTF{336cf6d51c9d9774fd37196c1d7320ff
```

5.2 Purple Thing

What could be inside this seemingly innocent thing?

If we do a Signature Analysis on this, we get -

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 780 x 720, 8-bit/color RGBA, non-interlaced
41            0x29            Zlib compressed data, best compression
153493        0x25795         PNG image, 802 x 118, 8-bit/color RGBA, non-interlaced
```

Looks like we have another PNG image inside this image, which is not a usual case, right? Let's dive more into it and do an Entropy Analysis.

```
DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
1024          0x400           Rising entropy edge (0.969866)
153600        0x25800         Rising entropy edge (0.963949)
156672        0x26400         Falling entropy edge (0.731285)
158720        0x26C00         Rising entropy edge (0.977674)
163840        0x28000         Falling entropy edge (0.613432)
```

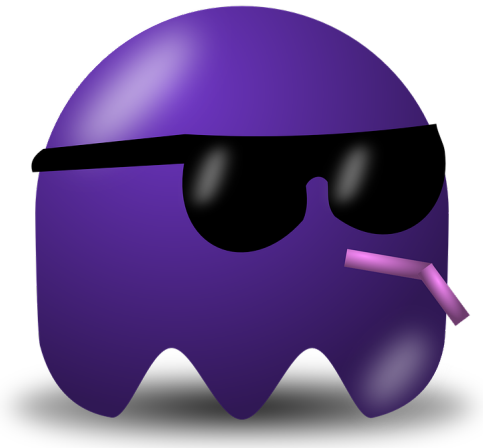


Figure 21: CTF Demo - PurpleThing

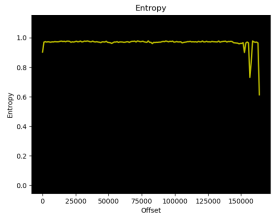


Figure 22: PurpleThing - Entropy Analysis

Logically, a png file like this should not have multiple risings and fallings. There must be another png inside this one!

Now if we run $binwalk - e$ on this, they only extract the zlib compressed data inside the png file that refers to the first png. How can we get the second png?

Let's run $binwalk - e - -dd = ".*"$ on it. It will extract and when matching the common files, they save that file.

This will give us the second PNG file, and when we open it, we get the flag.