Forensics from the Reverse-Engineering perspective

Reverse Engineering – Forensics from the Reverse-Engineering perspective
Sergi Àlvarez aka pancake

Forensics from the RE point of view

6. Forensics seen as a reverse engineer (3h)

Theory

- ~~Carving, magic headers, data recovery~~
- Binwalk... let's re-implement this in r2
- ~~Analyzing data structures (kaitai)~~
- Mounting filesystems, understanding routers memory layout (XIP, NAND…)
- Search capabilities of r2
- ~~Entropy calculations~~

Practice

- Extract binaries from a memory dump, from nand and from firmware
- ~~Optimizing the carving times~~
- ~~Displaying contents with different formats, telescoping~~

Forensics?

What is Forensics?

- Forensics is the art of discovering what happened in a specific system by carving information and understanding the data structures left in there.
- Usually happens in a post-mortem state.
  This is, after a specific event has happened.
  - An intrusion
  - Data Leak
  - …
- We take a copy of disk, memory, network captures and then analyze them statically without interfering with the original system.

What's the relationship with RE?

- Most forensic tools have been developed after doing reverse engineering on disk or memory dumps.
- In most situations we will not have access to the source of the program generating a specific file we want to parse or carve for unallocated blocks ends up requiring some expertise that can't be achieved by just reading the docs.
- This usually refers to disk dumps and parsing file systems, but it extends to all the artifacts that can be extracted. Caches, logs, temporary files, registry, etc.
- Most of them are closed source and have no specs.

Use Cases for Forensics

The post-mortem analysis of computer systems has multiple applications in the real world, legal complains and law enforcement agencies.

- Legal cases mainly, to proof someone did something.
- Finding out what an intruder did after breaking into a machine.
- Analyzing a crash state to understand what is causing a problem.
- Recovering deleted files.

Acquisition

Acquisition

The process of capturing a copy of a medium or stream into a file for the purpose of later analysis.

Sources

- The way we take the data from the system of interest is the most important part of the acquisition process.
- Also, we need to know which are the sources that we can take and which data we can find in there.
- The information is not stored in the same way in RAM or disk, or even a cache; each storage optimizes access times, resources needed, redundancy, etc.
  - So it uses different data structures and organization that we, as forensic analysts or reverse engineers must understand.

Disk Dump

- Disk is the most common source of information when performing a forensic analysis.
- Data survives reboots, and deleted files are just marked as removed, not wiped.
- We can use tools like dd to dump block device contents into a file.
- Block devices are named like this because data is stored in blocks, sectors, this is, the hardware only allows to read and write blocks of a specific size on aligned addresses.

Disk Dump Hashing

- While dumping the data is interesting to generate a checksumming file in order to verify that the contents have not been manipulated.
- If we compute the hashing we can do incremental checks and fail fast in case the disk has been modified.

```
$ rahash2 -a md5 -B -b 4M /dev/sda
```

Exercise: Disk Dump Hashing

- While dumping the data is interesting to generate a checksumming file in order to verify that the contents hasn’t been manipulated.
- If we compute the hashing by blocks we can perform partial checksums which are faster and allows us to quickly identify which blocks has changed if we have more than one dump.
- Write a shell script to dump the contents of a file system and perform a block hash verification to identify which parts of the device has been modified.

Partitions & FileSystems

Storage

FileSystems are software abstractions to provide a directory tree of files with attributes, permissions and contents on top of a memory device.

- Hard drive, aka block
- Flash memory, NAND...
- RAM, volatile memory (tmpfs …)

Each storage has its own characteristics, like different alignment restrictions.
Different speed performance on read or write operations require implementing caches or journals.

Partition Tables

There are different ways to partition a disk in order to have multiple filesystems in there.

This can be useful to have a home partition shared between different operating systems installed in the same system or just split the disk to organize the data.

There are different partition tables, each one with different characteristics and target operating system.

Partition Tables Types

The most common partition table format on PCs is MBR.

- But there's also EBR, GPT, BSD Disklabels, …
- https://en.wikipedia.org/wiki/Partition_table

We can use the mp command of r2 to list the partition types, offsets and sizes. So we can later dump or mount the partitions inside the disks using this informations.

- Mainframes don't have partition tables, only hardcoded offsets

FileSystems

There are many file systems out there, the most common ones are:

- Apple (HFS+, APFS)
- Windows (FAT, NTFS)
- Linux (EXT3, JFS, XFS, ..)
- Other (ISO9660, UDF, …)
- Routers (jffs, squashfs, yaffs2, ubifs)

FileSystems

Each file-system have its own characteristics to fit the needs that aims to cover:

- Compression
- Optimized for read operations
- Optimized for servers and raid setups
- Designed for small or large storages.

FileSystems Data Structures

- Header Magic
- Super blocks
- Sectors

Unallocated Area

- This concept defines the regions of the disk that are not referenced by any node of the filesystem.
- This is where we may want to search for deleted files.
- SleuthKit comes with the `blkls` utility that dumps the contents of the unallocated space from a known partition.

https://wiki.sleuthkit.org/index.php?title=FS_Analysis

Filesystem Journals

- A common technique that many file-systems use to avoid data corruption.
- Stored a linear log of changes that must be applied in an atomic and safe way.
- Those logs are saved in a hidden file, some file systems allow to access them, for example in ext3, you can access this file when mounting it as ext2

Userland Filesystems

- Traditionally most filesystems are implemented in kernel level, but nothing blocks you from implementing them in userland.
- FUSE is an api to implement userland filesystems
- Virtual Filesystem inspired by GNU/Hurd

Other userland filesystems:

- dosfstools
- sleuthkit
- r2 uses fork of the GRUB code.

Mounting FileSystem with r2

Use the ‘m’ command to mount a specific file system inside r2.

Extract a file from there using r2 or sleuthkit.

SleuthKit

SleuthKit

SleuthKit is an open-source and command-line tooling for forensics.

```
$ pacman -Ql sleuthkit | grep /bin/
```

- Supports raw and ewf images (-i raw, -i ewf)
- Parse partition tables (mmls)
- Enumerate inodes (ils)
- List filesystem contents (fls)
- FileSystem data units (blkls)
- Journal Images (jls)

What can we do?

Extract all the unallocated area into a single linear file for better carving deleted files.

- Help on identifying file-system corruption bugs.
- Recover data from the Journal image.
- Carve for a specific hex-string (using sigfind).
- Extract contents of unlinked inodes.

But.. wasn’t r2 designed for forensics

Yes, r2, started as a forensics tool

- Open large files, find patterns, extract results.
- Mount filesystem and enumerate partition tables.

Apart from that there are few things you can do except:

- Get the offset of the inode from a file.
- Get the file corresponding to an offset.

Ideally r2 should be integrated with sleuthkit.

Other plans for r2 about forensics

- Support volatility
  - Integrate volatility info inside r2
- Support userland FUSE in RFS
  - Reuse 3rd party userland implementations
- Support more file-systems (mainly for routers)
  - SquashFS
  - JFFS2
  - YAFFS
  - …
-

RAM

Memory Dump

- RAM memory is a fast and volatile storage
- Can be only dumped when the system is on
- Contents change constantly

There are some techniques to dump memory with the system shutted down:

- Requires liquid nitrogen
- Kernel module hang the system until done

Hibernate / Suspend

- It is also possible to dump the contents of the RAM into disk by hibernating the system.

- Suspend just hangs the system and keeps powering the RAM with voltage, so it will keep consuming:

  - In this state it is possible to freeze the slots with nitrogen.
  - Replace the ram into another system with a custom boot process that dumps the ram to disk.
  - It is proven that the data corrupts over time and it is possible to recover some of the data.
  - The low temperatures slow downs the process.

Swap

- Modern operating system have the ability to use the hard drive as an extension of the RAM.
- This is called swap and can be done in a separate partition or just use a restricted file in the filesystem.
- Windows and macOS use a file and Linux usually a partition.
- This is a good place to find user data when the system is low in memory.

Hibernate

- This operation must be implemented by the kernel.
- Dumps the ram and kernel state to disk info a file.
- This file is loaded by the kernel at boot time and restores the system in the previous state.

In macOS:

- `/private/var/vm {swapfile*,sleepimage}`

Virtual Machine

In the case of running the system in a virtualized environment we can get a copy of the RAM of the system into a file.

- qemu
- vbox
- vmware
- xen
- kvm
- ...

Volatility

Volatility

- Tool to analyze memory dumps
  - Supports different Profiles (ships only defaults for Windows)
  - Need to create your owns for Linux, and macOS
- Memory dumps can be saved in different formats
  - VirtualBox uses a huge elf64 to store this
  - QEMU, VMWare, Lime
  - See Address Spaces in `vol.py --info | less` for more
- Each profile have some actions associated (Linux, Windows, Mac), same goes for containers.

Exercise: Dump VirtualBox ram

- `VBoxManage debugvm radare2 dumpvmcore --filename=/tmp/vm.core`

```
$ python vol.py  -f /tmp/vm.core vboxinfo
Volatility Foundation Volatility Framework 2.6
Magic: 0xc01ac0de
Format: 0x10005
VirtualBox 5.2.2 (revision 119230)
CPUs: 4

FileOffset Memory Offset Size
    0x8bec            0x0 0x80000000
0x80008bec    0xe0000000   0xc00000
0x80c08bec    0xf0400000   0x400000
0x81008bec    0xf0800000     0x4000
0x8100cbec    0xffff0000    0x10000
$
```

Volatility

Create a Linux VM

- nokaslr in boot parameters
- `make -C tools/linux # module.dwarf`
- System.map is generated by `mksysmap` script in `/lib/modules*`
- Create a zip with those 2 files and place it in `volatility/plugins/overlay/linux/ArchLinux.zip`
- Select the `--profile LinuxArchLinuxx64`
  - Note ‘Linux’ and ‘x64’ are added automatically
  - So probably find a better name next time

Volatility

```
python vol.py -info | grep linux
```

- We get a list of all the commands to use with this memory dump
- Test some, like listing processes

R2K

r2k

- r2 plugin and a kernel module
- Only for Linux and Windows atm
  - macOS support coming soon
- Allow r2 to read and write physical, linear or process memory by using a kernel extension.
- Can read control registers.
- Developed by panda and oscar with nighterman mentoring during last GSoC. Thanks SkUaTeR.

What can we do with r2k?

- Inspect kernel memory, and perform hotpatches.
- No special protections are done when you modify the kernel memory.
- Inspect other processes memory without raising the trace bit, so the process will not know that is being observed.
- Dump kernel, linear or physical memory to disk.
- Disassembler kernel code to better understand crash logs.

Exercise: Got Root?

- Compile and install r2k
  - `r2pm -i r2k`
- `r2 r2k:///dev/r2k`

Volatility give us the address of the process structure in RAM.
We can use this pointer from r2 to:

- find the address of the pointer to the credentials structure
- do the same for pid 1
- `memcpy 0x80480 0x804 32`

Network

Network Capture

- Capturing data from the network is a very important source of information when trying to understand what happened during the event of interest.

- Network traffic can be only captured in real-time, so it is important to have a constant device sniffing data in order to keep track of all the packets.

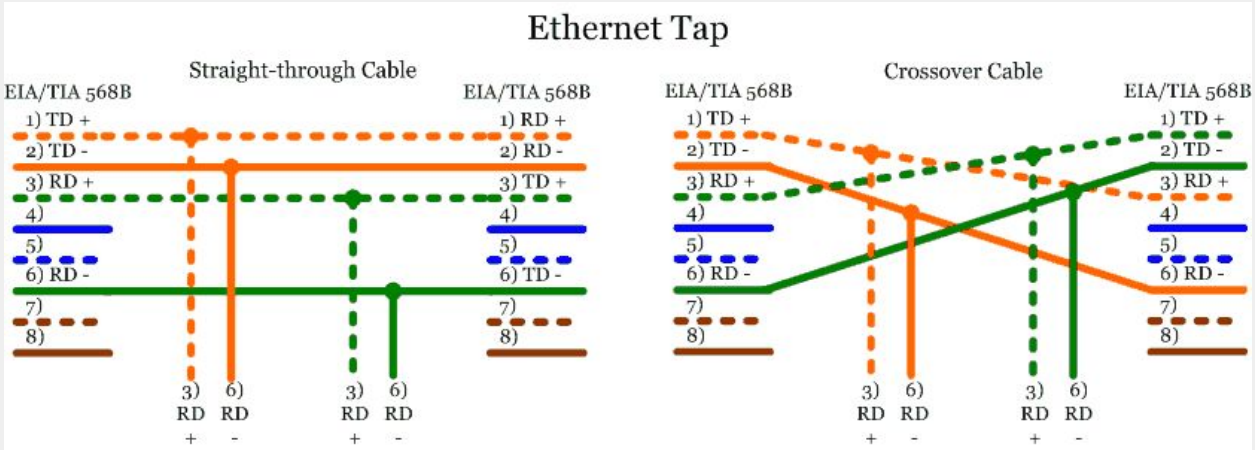
- The most common format is PCAP

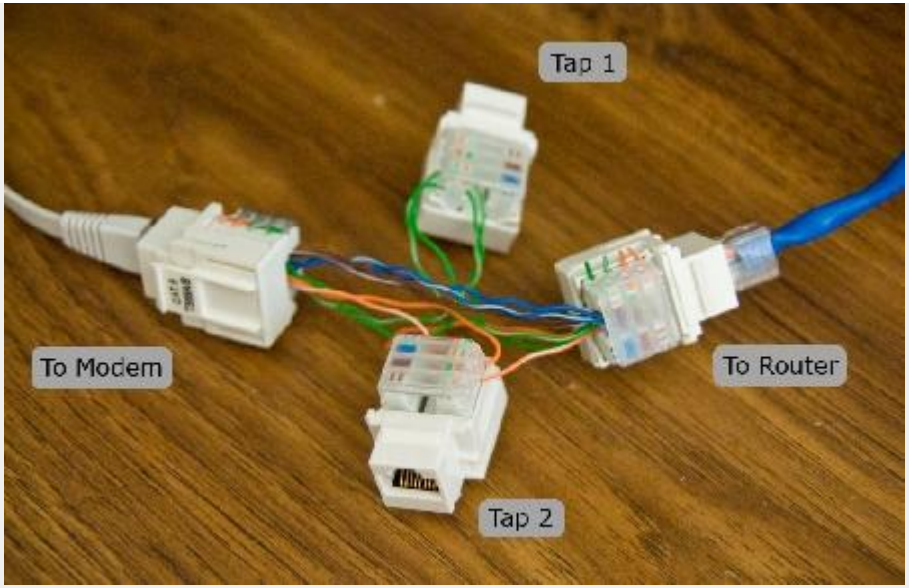
Active vs Passive

There are different ways to capture network traffic:

- Inside the target machine.
- From a network device in front of the target machine.
- From the DMZ port:
  - replicates traffic from all ports
- Passive Network Tap
  - https://hackaday.com/2008/09/14/passive-networking-tap/

Passive Network Tap





Why is Active Sniffing Wrong?

- The logs must be later transferred for analysis.
- Extra noisy traffic that must be filtered.
- Poisoned ARP.
- Capturing increases CPU usage and affects network traffic processing performance.
- Storing the capture requires disk storage which is very limited in many cases (routers …)
- Can be detected
  - `nmap --script=sniffer-detect 192.168.114`
  - `detect_sniff`
  - https://null-byte.wonderhowto.com/forum/hiob-detecting-sniffing-device-network-0159181/

Network Capture

- The main problems when analyzing network data:
  - Reconstructing packet flow
  - Decrypting encrypted data
  - Finding data fragmentation

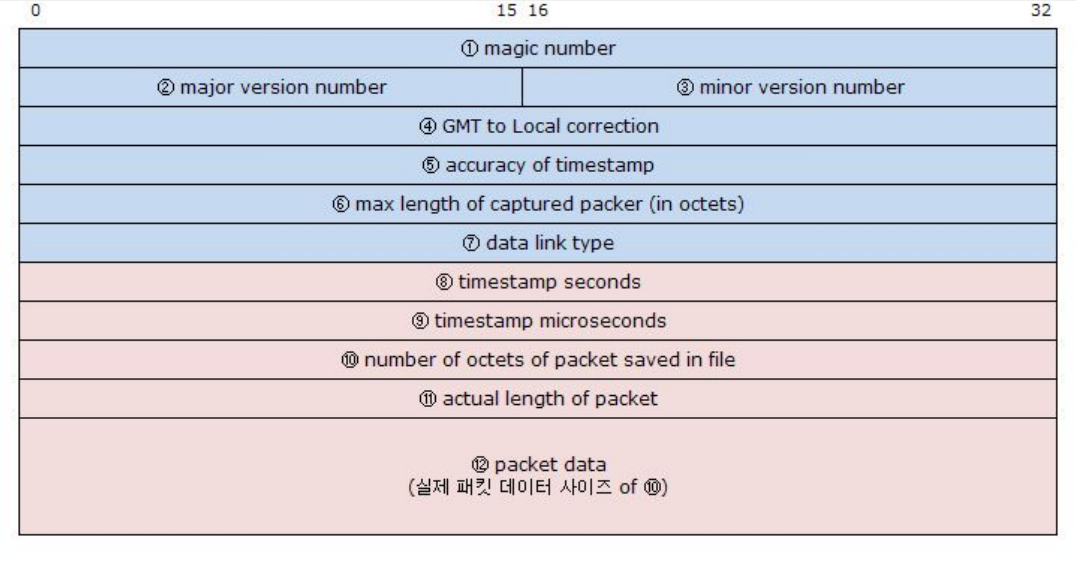
- Tools
  - Xplico
  - tcpdump
  - ngrep
  - Wireshark
  - ...

Exercise: Network Capture

- Create a pcap file
  - Sniff our network interface
- Use `rabin2 -z` or `strings`
  - What's the difference?
- Use `tcpdump -r` or `ngrep -r`
- Explain how network filters work
  - `man tcpdump`

The PCAP format

- PCAP stands for “Packet Capture”
- It is a very simple file format to contain all the packets captured from the network.
- Some CTFs use corrupted PCAPs to crash analyzer tools



Exercise: Parsing PCAP

In this practice we will learn how to use r2 pf and t commands

- `r2pm -i pcap`
- `x @@ sym*`
- `to t/pcap.h`
- `.ts pcap_file_hdr @ 0`
- `.ts pcap_pktrec_ipv4 @@= `f~IPV4[0]``
- `.ts*pcap_file_hdr`

Carving

Carving

Having a dump of a disk, memory, network, find inside it is possible to extract

- Finding known magic numbers or headers in a raw dump.
- Guess the size of its contents by doing assumptions.
- Memory structures are not the same as in disk:
  - ELF/PE/… can’t be dumped to disk and expect it to work
- Useful for forensics to recover deleted data:
  - Copied multiple times
  - Some of those copies are corrupted
  - Dump them all and find the right copy

Identifying file formats

There are different ways to identify the type of a file.

- File extension
  - Used mainly by windows users.
- MimeType metadata
  - Used when in transport (http, mail, …)
- File header
  - Assume some magic numbers to be there
- File contents

Magic Headers

- Why are they called magic?

Because those constant numbers are defined without any reason behind, just taken arbitrary numbers assuming no one else will use the same

- Do you know file program from UNIX?
  - `man file`
- How are those signatures defined?
  - `vim libr/magic/d/*`
- Yara Rules

UNIX FILE

The tool ‘file’ takes a database file header definitions and try to guess which of them match in the target files or stdin data feeded into the tool.

- `man file`
- `libmagic`
- `r2/libr/magic`

Magic File Definitions

```
#------------------------------------------------------------------------------
# elf:  file(1) magic for ELF executables
#
# We have to check the byte order flag to see what byte order all the
# other stuff in the header is in.
#
# What're the correct byte orders for the nCUBE and the Fujitsu VPP500?
#
# updated by Daniel Quinlan (quinlan@yggdrasil.com)
0       string          \177ELF         ELF
>4      byte            0               invalid class
>4      byte            1               32-bit
>4      byte            2               64-bit
>5      byte            0               invalid byte order
>5      byte            1               LSB
>>16    leshort         0               no file type,
!:mime  application/octet-stream
>>16    leshort         1               relocatable,
!:mime  application/x-object
>>16    leshort         2               executable,
!:mime  application/x-executable
>>16    leshort         3               shared object,
!:mime  application/x-sharedlib
# Core handling from Peter Tobias <tobias@server.et-inf.fho-emden.de>
# corrections by Christian 'Dr. Disk' Hechelmann <drdisk@ds9.au.s.shuttle.de>
>>16    leshort         4               core file
!:mime  application/x-coredump
# Core file detection is not reliable.
#>>>(0x38+0xcc) string  >\0             of '%s'
#>>>(0x38+0x10) lelong  >0              (signal %d),
>>16    leshort         &0xff00         processor-specific,
>>18    leshort         0               no machine,
>>18    leshort         1               AT&T WE32100 - invalid byte order,
>>18    leshort         2               SPARC - invalid byte order,
```

Polyglot Files

- Sometimes the magic header gives some room to allow more than one file format to be compatible with it.
- Zip, PDF, PHP
- PoC || GTFO ezines are a good example of this

Ange Albertini (corkami) did some neat work on this

- https://en.wikipedia.org/wiki/Polyglot_(computing)
- https://github.com/corkami/pocs/tree/master/poly
- https://github.com/Talanor/Pylyglot

False Positives

- Usually when carving with magic signatures on some corrupted storages like RAM dumps or fragmented ones in PCAP we may find false positives.
- We have to add a filter logic to reduce the amount of results.
- We can also remove signatures to speedup the search and reduce false positives.

Exercise: Parse some headers

- Understand how to identify
- Open JPG, ZIP, ELF, MP3, PDF, Java Class
- Find a MAGIC that we can assume its unique

Yara

Yara

- Open Source tool that takes some yara rules (signatures)
- Support modules for different file formats or backends to extract and compare information on different sources.
- Mainly used to classify malware

Yara Rules

```
rule apt_backspace
{

    meta:
        description = "Detects APT backspace"
        author = "Bit Byte Bitten"
        date = "2015-05-14"
        hash = "6cbfeb7526de65eb2e3c848acac05da1e885636d17c1c45c62ad37e44cd84f99"

    strings:
        $s1 = "!! Use Splice Socket !!"
        $s2 = "User-Agent: SJZJ (compatible; MSIE 6.0; Win32)"
        $s3 = "g_nAV=%d,hWnd:0x%X,className:%s,Title:%s,(%d,%d,%d,%d),BOOL=%d"

    condition:
        uint16(0) == 0x5a4d and all of them
}
```

Run Yara on some binaries.

- Take different metrics from a library
- Identify which binaries has been statically linked with this library.
- Use Yara for this

BinWalk

Binwalk

- Most famous tool for carving firmwares to identify its pieces.
- It is designed to analyze firmwares without knowing their formats.
- May help us to get some pointers to understand the headers structures.
- There are other carving tools like photorec

Using binwalk

- `sudo pip install binwalk`

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
1761488       0x1AE0D0        PNG image, 1324 x 86, 8-bit/color RGB, non-interlaced
4400532       0x432594        Microsoft executable, MS-DOS
26961610      0x19B66CA       MySQL ISAM compressed data file Version 5
58616226      0x37E69A2       StuffIt Deluxe (data): TDh
69812195      0x4293FE3       Microsoft executable, MS-DOS
73582463      0x462C77F       MySQL ISAM compressed data file Version 1
76578694      0x4907F86       Microsoft executable, MS-DOS
78599156      0x4AF53F4       Microsoft executable, MS-DOS
86997623      0x52F7A77       Microsoft executable, MS-DOS
87019363      0x52FCF63       Microsoft executable, MS-DOS
87078145      0x530B501       Microsoft executable, MS-DOS
90826658      0x569E7A2       Microsoft executable, MS-DOS
97633313      0x5D1C421       Microsoft executable, MS-DOS
```

Using binwalk

- Binwalk supports magic files (-m)
- Use -e to extract all the findings
- Recursive scanning with -M
- Compute entropy with -E

In addition it supports some binary diffing features

- -w, -W

Searching with r2

- R2 is a very versatile and powerful hexadecimal editor.

- Understanding how to use the / command can help us in many use cases
- Checking out all the options under the search. eval
  - search.align
  - search.in=?
  - search.contiguous=?
  - search.overlap
- How to specify multiple keywords to search
  - Zignatures?
  - Magic?

Implementing our own binwalk with r2

- Which language use here?
  - Shellscript?
  - NodeJS?
  - Python?
  - r2 scripting?
- Features we are interested:
  - Specify one or more files
  - Find all known types
  - Try to guess the hit size
  - Dump results
  - Output logs

Visualization

Large File Visualizations

One of the most important problems when dealing with big files is to try to get an overview of it, understanding what are the contents and the structure of the file.

There are multiple ways to do that, probably the first one that come to our minds is defrag.exe

Large File Visualizations

So, assuming each square shows information about a portion of the file and the color give us information about its contents, we can use different algorithms to compute that.

The bottleneck is the io; the fewer bytes we read the faster.

- Compute a value with the contents of each block
- Specify a size for the blocks.

Entropy Calculations

Different values of entropy will tell us about the type of contents of each block.

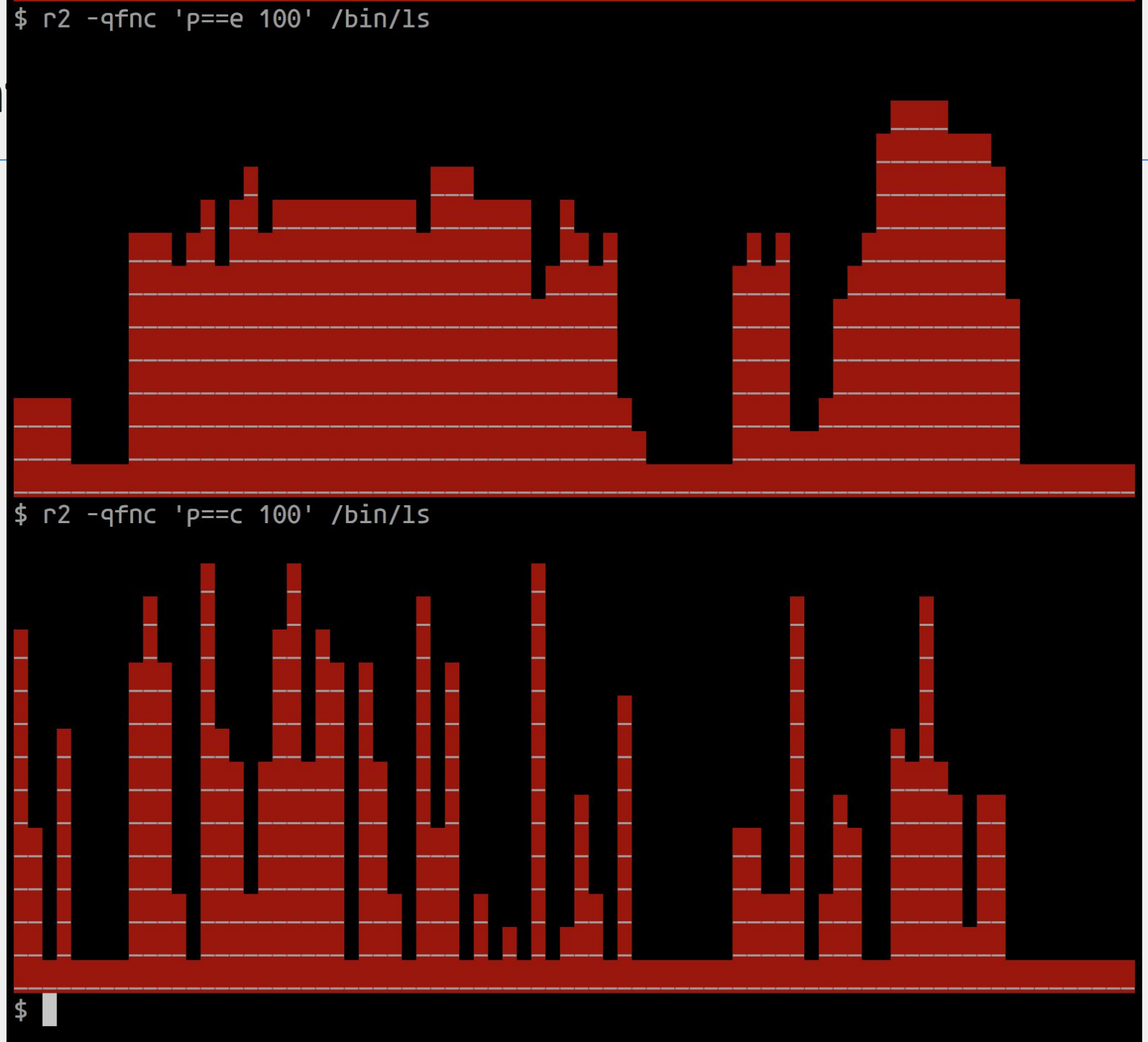
- It is plaintext?
- It is compressed?
- It is code or data?

Entropy

- The entropy is a value that represents the amount of distribution of possible values for each block

```
R_API double r_hash_entropy(const ut8 *data, ut64 size) {
        if (!data || !size) {
                return 0;
        }
        ut64 i, count[256] = {0};
        double h = 0;
        for (i = 0; i < size; i++) {
                count[data[i]]++;
        }
        for (i = 0; i < 256; i++) {
                if (count[i]) {
                        double p = (double) count[i] / size;
                        h -= p * log2 (p);
                }
        }
        return h;
}
R_API double r_hash_entropy_fraction(const ut8 *data, ut64 size) {
        return size ? r_hash_entropy (data, size) / \
                log2 ((double) R_MIN (size, 256)) : 0;
}
```

En



DataStructs;

Data Structures

Computers store information in binary form using data structures, depending on the needs, those data structures can be different, we may want to understand this in order to be able to parse or guess the contents of an unknown data structure.

- Alignment depends on storage (ram, nand, …)
- Type of data (int, uint, void*)
- Size, usually linked to the type.
- Name, to ease human’s life
- Arrays
- Endianness

Data Structures in C

C is probably the best language to test and understand how data is stored in memory.

- But other languages provide decent apis to process them.
- Python pack/unpack inspired in Perl ones is one of them.
- R2's RRead and RBuffer apis provide the primitives as well.
- Basically, any language providing ffi interop may provide the tooling to parse.

Data Structures in C

```
struct foo {

  uint32_t foo;  // 32 bit + 4 byte
alignment padding on 64

  const char *bar; // 32 or 64

  short cow; // 16 bit with padding to
fill 4 or 8

}
```

Packed Structures and Field Alignment

By default, compilers will fill the structures by aligning the fields and adding padding. There are some reasons for this

- Most MMUs work better when reading alignment memory
  - Performance, avoid unnecessary reads, lazy processors
- Not all instructions fetch memory with the same circuitry
  - fpu, gpr, mmx, …
- Atomicity (locking unaligned memory is not working)

https://www.ibm.com/developerworks/library/pa-dalign/

http://www.catb.org/esr/structure-packing/

Structure reordering

Compilers use to reorder the fields of the structure in order to fill the gaps and reduce its size.

This is done when the optimization flag is set.

Take this structure and optimize it by reordering the fields.

```
struct Unordered {

  short a; int b;short

};
```

Data Telescoping

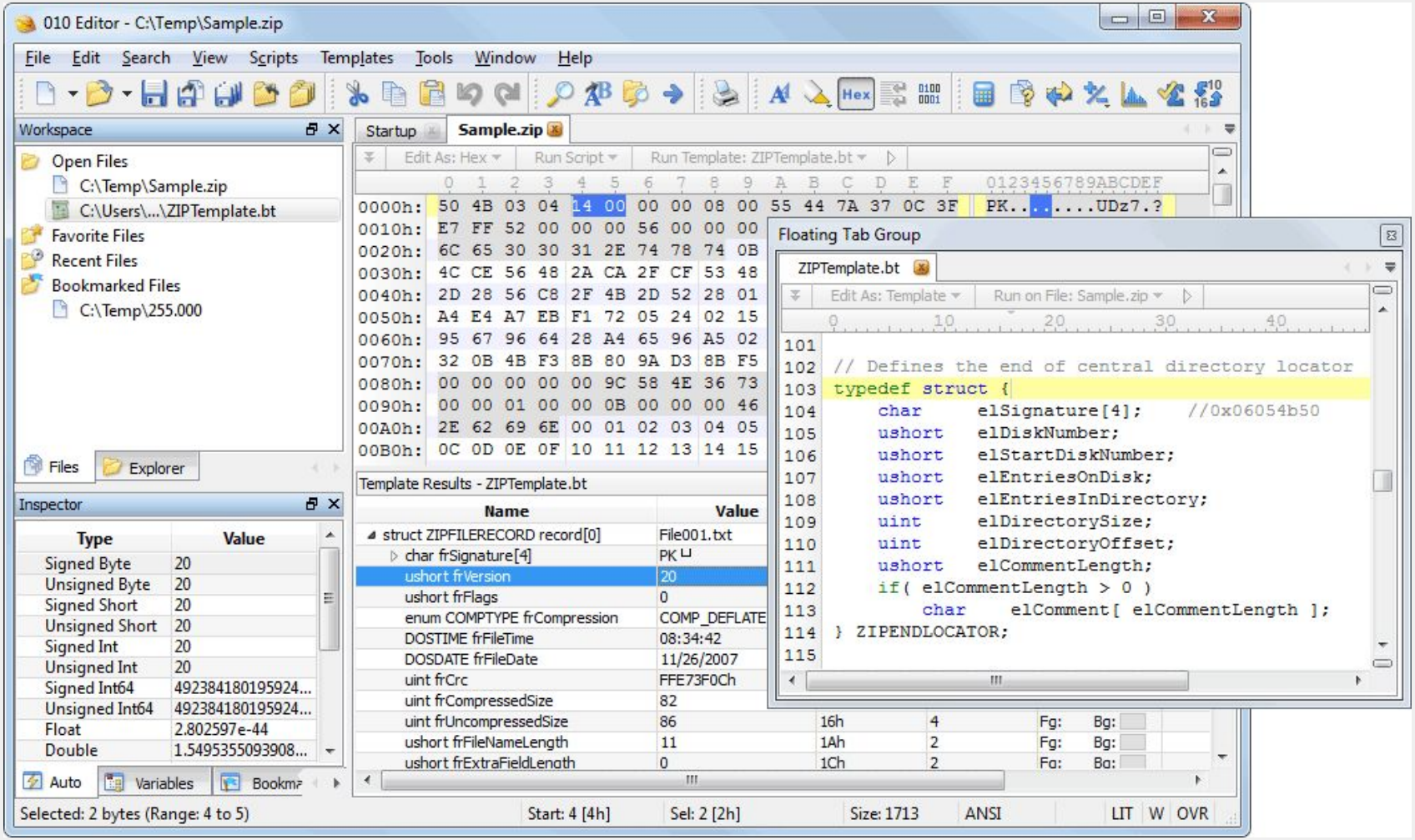
- When interpreting data, sometimes we will find pointers stored in memory, and we want to follow them to see what's pointing to.
- This is called telescoping.
- gdb-peda and r2 do that
- If it is pointing to a struct we want to show its contents and fields.
- Some pointers may lead to pointers that will result in infinite loops. So we have to define a maximum depth.

Other Tools

There are some tools that may help us to inspect data and understand better its contents.

- 010 editor
  - Comercial hexadecimal editor with support for binary templates
- Kaitai
  - OpenSource project that implements similar templating engine.

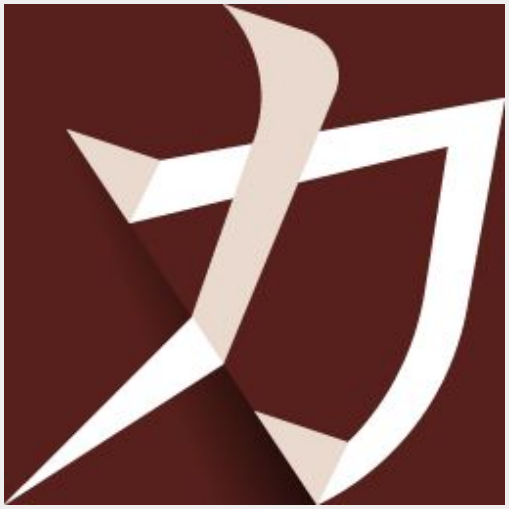
010 editor


010 Editor - C:\Temp\Sample.zip
File Edit Search View Scripts Templates Tools Window Help
Workspace
Open Files
C:\Temp\Sample.zip
C:\Users\...\ZIPTemplate.bt
Favorite Files
Recent Files
Bookmarked Files
C:\Temp\255.000
Files
Explorer
Inspector
Type
Value
Signed Byte 20
Unsigned Byte 20
Signed Short 20
Unsigned Short 20
Signed Int 20
Unsigned Int 20
Signed Int64 492384180195924...
Unsigned Int64 492384180195924...
Float 2.802597e-44
Double 1.5495355093908...
Auto
Variables
Startup
Sample.zip
Edit As: Hex
Run Script
Run Template: ZIPTemplate.bt
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0000h: 50 4B 03 04 14 00 00 00 08 00 55 44 7A 37 0C 3F PK..........UDz7.?
0010h: E7 FF 52 00 00 00 56 00 00 00
0020h: 6C 65 30 30 31 2E 74 78 74 0B
0030h: 4C CE 56 48 2A CA 2F CF 53 48
0040h: 2D 28 56 C8 2F 4B 2D 52 28 01
0050h: A4 E4 A7 EB F1 72 05 24 02 15
0060h: 95 67 96 64 28 A4 65 96 A5 02
0070h: 32 0B 4B F3 8B 80 9A D3 8B F5
0080h: 00 00 00 00 00 9C 58 4E 36 73
0090h: 00 00 01 00 00 0B 00 00 00 46
00A0h: 2E 62 69 6E 00 01 02 03 04 05
00B0h: 0C 0D 0E 0F 10 11 12 13 14 15
Template Results - ZIPTemplate.bt
Name
Value
struct ZIPFILERECORD record[0] File001.txt
char frSignature[4] PK
ushort frVersion 20
ushort frFlags 0
enum COMPTYPE frCompression COMP_DEFLATE
DOSTIME frFileTime 08:34:42
DOSDATE frFileDate 11/26/2007
uint frCrc FFE73F0Ch
uint frCompressedSize 82
uint frUncompressedSize 86
ushort frFileNameLength 11
ushort frExtraFieldLength 0
16h 4 Fg: Bg:
1Ah 2 Fg: Bg:
1Ch 2 Fg: Bg:
Floating Tab Group
ZIPTemplate.bt
Edit As: Template
Run on File: Sample.zip
101
102 // Defines the end of central directory locator
103 typedef struct {
104 char elSignature[4]; //0x06054b50
105 ushort elDiskNumber;
106 ushort elStartDiskNumber;
107 ushort elEntriesOnDisk;
108 ushort elEntriesInDirectory;
109 uint elDirectorySize;
110 uint elDirectoryOffset;
111 ushort elCommentLength;
112 if(elCommentLength > 0)
113 char elComment[elCommentLength];
114 } ZIPENDLOCATOR;
115
Selected: 2 bytes (Range: 4 to 5)
Start: 4 [4h]
Sel: 2 [2h]
Size: 1713
ANSI
LIT W OVR

Kaitai Project

Declarative language used for describe various binary data structures.

Have support for many programming languages, and also a frontend



r2: pf command

Radare2 provides a way to represent contents of data using a “format string” that represents the contents of the memory.

- Supported formats can be listed with pf??
- pf subcommands are documented in pf?

pf format strings are a state machine that support multiple modifiers to each basic type:

- endianness
- pointer dereferences
- skip N bytes

Exercise: Using pf in r2

- Demo

Searching

Searching

Many times when doing Forensics we will need to search for some specific keywords in order to find documents, files containing them.

Those search are performed in the whole disk, but having tools like sleuthkit to dump, or specify those ranges.

Encodings

The main problem when dealing with keywords is that there are several ways to represent the same text and depending on the storage, or file format where it was contained it will be encoded differently.

- utf8
- widechar (utf16)
- utf32 (little or big endian)
- Latin1 (iso8859-1)
- Base64
- Gzip
- ...

Compressions

There are several different compression algorithms, and searching for compressed text in different encodings requires an inverse search.

This is. Find all compressed streams in the media, uncompress and search with all the encodings on the deflated buffer.

Searching for a Hash

When reverse engineering file formats we may find in many situations a hash of a region, used to verify that is not corrupted.

This is done by the /h command in r2

- Computes the $Hash for all offsets with a fixed $Length
- Stops when finding the offset and shows the result

Searching for Values

As we have learned, numbers can be stored in different endians and sizes. We can search for the hex-pairs with different encodings, but we should know the way the values are encoded for the specific target.

- 32 or 64 bit (4 or 8 bytes)
- Little or Big endian
- In range
- A reference in code

Find Firmware RAM references

- Open the MD380 firmware

- Use /v4 and /V4
- Use /r? to find references

What's a Hash / Checksum?

Hashes and checksums are different uses for the same concept. (See rahash2)

- CRC32
- MD4 / MD5
- SHA1
- SHA256
- XORPAIR
- LUHN
- XXHASH
- ADLER32

Optimizing Search Times

There are different ways to improve the performance, and we have already learned some of them when doing the magic search.

- Change block size
- Define alignment
- Disable contiguous hits

Bear in mind that some search algorithms are slower than others. For example: code analysis search, regex search, binary mask...

Multiple Keyword Search

- r1 had /k and /m to specify keywords and masks
- in r2 the API supports defining multiple keywords
  - You can’t do that from the command prompt yet.
  - It is possible with rafind2 from the command-line
  - Suggestions and improvements are welcome
- Feel free to join the community and suggest your improvements!

Diffing Data

Diffing Data

The last feature related to forensics and data analysis that we will explain today is to look for differences between two sources.

Depending on the source of data we will want to analyze those changes using different tools or representing the data using different renderers.

- The most known tool is GNU diff

Diffing Data

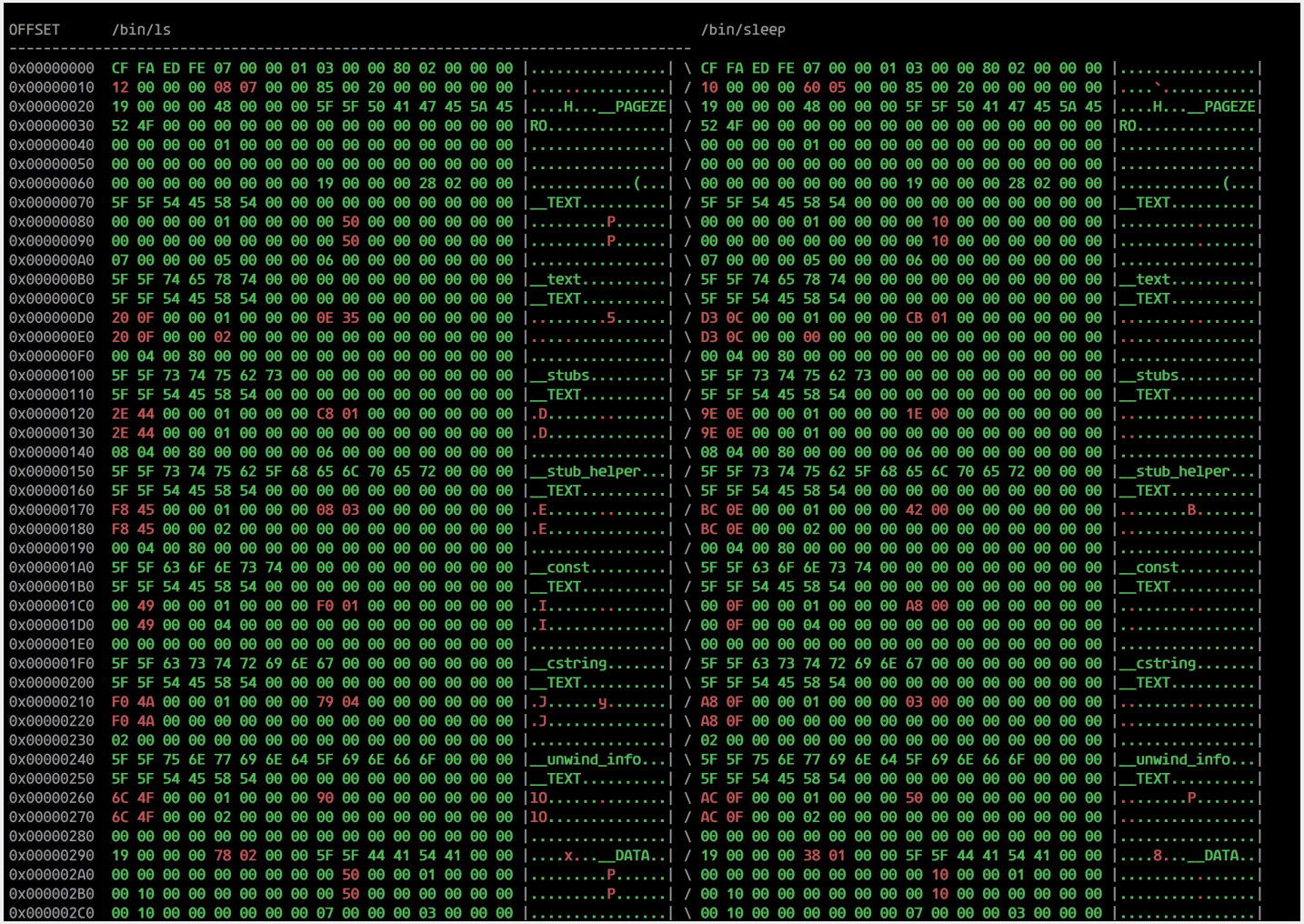
There are many other tools that focus on binary diffing, so you don’t have to hexdump | diff -u

- binwalk -W
- bdiff
- radiff2 (and the r2 c command)

radiff2 -x

```
  offset     0 1 2 3 4 5 6 7 01234567    0 1 2 3 4 5 6 7 01234567
0x00000000  cffaedfe07000001 ........   cffaedfe07000001 ........
0x00000008  0300008002000000 ........   0300008002000000 ........
0x00000010! 1000000060050000 ....`...   1000000000060000 ........
0x00000018  8500200000000000 .. .....   8500200000000000 .. .....
0x00000020  1900000048000000 ....H...   1900000048000000 ....H...
...
0x00000088! 0010000000000000 ........   0020000000000000 . ......
0x00000090  0000000000000000 ........   0000000000000000 ........
0x00000098! 0010000000000000 ........   0020000000000000 . ......
0x000000a0  0700000005000000 ........   0700000005000000 ........
0x000000a8  0600000000000000 ........   0600000000000000 ........
...
0x000000d0! d30c000001000000 ........   760c000001000000 v.......
0x000000d8! cb01000000000000 ........   8a0c000000000000 ........
0x000000e0! d30c000000000000 ........   760c000000000000 v.......
0x000000e8  0000000000000000 ........   0000000000000000 ........
0x000000f0  0004008000000000 ........   0004008000000000 ........
...
0x00000120! 9e0e000001000000 ........   0019000001000000 ........
0x00000128! 1e00000000000000 ........   0801000000000000 ........
0x00000130! 9e0e000001000000 ........   0019000001000000 ........
0x00000138  0000000000000000 ........   0000000000000000 ........
0x00000140  0804008000000000 ........   0804008000000000 ........
...
0x00000170! bc0e000001000000 ........   081a000001000000 ........
0x00000178! 4200000000000000 B.......   c801000000000000 ........
```

binwalk -W



```
OFFSET      /bin/ls                                                         /bin/sleep
--------------------------------------------------------------------------------
0x00000000  CF FA ED FE 07 00 00 01 03 00 00 80 02 00 00 00 |................| \ CF FA ED FE 07 00 00 01 03 00 00 80 02 00 00 00 |................|
0x00000010  12 00 00 00 08 07 00 00 85 00 20 00 00 00 00 00 |..............  | / 10 00 00 00 60 05 00 00 85 00 20 00 00 00 00 00 |....`.........  |
0x00000020  19 00 00 00 48 00 00 00 5F 5F 50 41 47 45 5A 45 |....H..._PAGEZE| \ 19 00 00 00 48 00 00 00 5F 5F 50 41 47 45 5A 45 |....H..._PAGEZE|
0x00000030  52 4F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |RO............. | / 52 4F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |RO............. |
0x00000040  00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |................| \ 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| / 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000060  00 00 00 00 00 00 00 00 19 00 00 00 28 02 00 00 |............(...| \ 00 00 00 00 00 00 00 00 19 00 00 00 28 02 00 00 |............(...|
0x00000070  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| / 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x00000080  00 00 00 00 01 00 00 00 00 50 00 00 00 00 00 00 |.........P......| \ 00 00 00 00 01 00 00 00 00 10 00 00 00 00 00 00 |................|
0x00000090  00 00 00 00 00 00 00 00 00 50 00 00 00 00 00 00 |.........P......| / 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00 |................|
0x000000A0  07 00 00 00 05 00 00 00 06 00 00 00 00 00 00 00 |................| \ 07 00 00 00 05 00 00 00 06 00 00 00 00 00 00 00 |................|
0x000000B0  5F 5F 74 65 78 74 00 00 00 00 00 00 00 00 00 00 |__text..........| / 5F 5F 74 65 78 74 00 00 00 00 00 00 00 00 00 00 |__text..........|
0x000000C0  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| \ 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x000000D0  20 0F 00 00 01 00 00 00 0E 35 00 00 00 00 00 00 |.........5......| / D3 0C 00 00 01 00 00 00 CB 01 00 00 00 00 00 00 |................|
0x000000E0  20 0F 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |................| \ D3 0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000000F0  00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00 |................| / 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000100  5F 5F 73 74 75 62 73 00 00 00 00 00 00 00 00 00 |__stubs.........| \ 5F 5F 73 74 75 62 73 00 00 00 00 00 00 00 00 00 |__stubs.........|
0x00000110  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| / 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x00000120  2E 44 00 00 01 00 00 00 C8 01 00 00 00 00 00 00 |.D..............| \ 9E 0E 00 00 01 00 00 00 1E 00 00 00 00 00 00 00 |................|
0x00000130  2E 44 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |.D..............| / 9E 0E 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000140  08 04 00 80 00 00 00 00 06 00 00 00 00 00 00 00 |................| \ 08 04 00 80 00 00 00 00 06 00 00 00 00 00 00 00 |................|
0x00000150  5F 5F 73 74 75 62 5F 68 65 6C 70 65 72 00 00 00 |__stub_helper...| / 5F 5F 73 74 75 62 5F 68 65 6C 70 65 72 00 00 00 |__stub_helper...|
0x00000160  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| \ 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x00000170  F8 45 00 00 01 00 00 00 08 03 00 00 00 00 00 00 |.E..............| / BC 0E 00 00 01 00 00 00 42 00 00 00 00 00 00 00 |........B.......|
0x00000180  F8 45 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |.E..............| \ BC 0E 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000190  00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00 |................| / 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001A0  5F 5F 63 6F 6E 73 74 00 00 00 00 00 00 00 00 00 |__const.........| \ 5F 5F 63 6F 6E 73 74 00 00 00 00 00 00 00 00 00 |__const.........|
0x000001B0  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| / 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x000001C0  00 49 00 00 01 00 00 00 F0 01 00 00 00 00 00 00 |.I..............| \ 00 0F 00 00 01 00 00 00 A8 00 00 00 00 00 00 00 |................|
0x000001D0  00 49 00 00 04 00 00 00 00 00 00 00 00 00 00 00 |.I..............| / 00 0F 00 00 04 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| \ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001F0  5F 5F 63 73 74 72 69 6E 67 00 00 00 00 00 00 00 |__cstring.......| / 5F 5F 63 73 74 72 69 6E 67 00 00 00 00 00 00 00 |__cstring.......|
0x00000200  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| \ 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x00000210  F0 4A 00 00 01 00 00 00 79 04 00 00 00 00 00 00 |.J......y.......| / A8 0F 00 00 01 00 00 00 03 00 00 00 00 00 00 00 |................|
0x00000220  F0 4A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.J..............| \ A8 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000230  02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| / 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000240  5F 5F 75 6E 77 69 6E 64 5F 69 6E 66 6F 00 00 00 |__unwind_info...| \ 5F 5F 75 6E 77 69 6E 64 5F 69 6E 66 6F 00 00 00 |__unwind_info...|
0x00000250  5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........| / 5F 5F 54 45 58 54 00 00 00 00 00 00 00 00 00 00 |__TEXT..........|
0x00000260  6C 4F 00 00 01 00 00 00 90 00 00 00 00 00 00 00 |lO..............| \ AC 0F 00 00 01 00 00 00 50 00 00 00 00 00 00 00 |........P.......|
0x00000270  6C 4F 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |lO..............| / AC 0F 00 00 02 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| \ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000290  19 00 00 00 78 02 00 00 5F 5F 44 41 54 41 00 00 |....x..._DATA..| / 19 00 00 00 38 01 00 00 5F 5F 44 41 54 41 00 00 |....8..._DATA..|
0x000002A0  00 00 00 00 00 00 00 00 00 50 00 00 01 00 00 00 |.........P......| \ 00 00 00 00 00 00 00 00 00 10 00 00 01 00 00 00 |................|
0x000002B0  00 10 00 00 00 00 00 00 00 50 00 00 00 00 00 00 |.........P......| / 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00 |................|
0x000002C0  00 10 00 00 00 00 00 00 07 00 00 00 03 00 00 00 |................| \ 00 10 00 00 00 00 00 00 07 00 00 00 03 00 00 00 |................|
```

Diffing Data

The previous screenshot show how a plain hexdump binary diffing looks with radiff2 and binwalk.. but if we are trying to understand the changes between two different data structures, we have no real information about the fields and type of the structures.

- Solutions? improve existing tools
- Build our own tool on top of the current results based on r2 and pf to show the formatted data structures and then use the unified gnu diff to read the changes.

Creating Binary Patches

bdiff2 and radiff2 support, not just finding the differences between two binaries, but also creating binary patches that can be applied to one binary to get the other one or vice versa.

Let’s make a quick demo of this to understand how it works and how are structured those binary patches:

$ radiff2 -r A B

- Output is in r2 commands
  - w to write
  - r to resize or insert

```
$ radiff2 -dr /bin/sleep /bin/ls | grep ^r
File size differs 18080 vs 38688
r-1259 @ 0x00000000
r+1579 @ 0x00000000
r-7884 @ 0x000004ef
r+28181 @ 0x000004ef
r-4840 @ 0x000033b8
r+4831 @ 0x000033b8
$
```

Practice: Understanding a Patch

- Take two binaries and find what has changed
- What can we say about this patch?

Questions?