

# Digital Forensics for SSC Solvers

Daniel Kouril

EGI CSIRT

# Digital Forensics

- Methods to collect and analyze (digital) evidence
- Three basic phases
  - Data collection, Analysis, Reporting
  - Triage
- Various sources of information
  - Host, network
- Online (live) vs. Offline analysis
- [https://wiki.egi.eu/wiki/Forensic\\_Howto](https://wiki.egi.eu/wiki/Forensic_Howto)

# Triage – is there an incident or not?

- Minimize actions
  - Every contact leaves a trace
- Quickly examine the system
  - Looking for anomalies
  - Even minor things may matter
- If incident is confirmed- isolate services/machines
  - Proceed to contain incident

# Starting investigation

- Leif Nixon's cup of tea/coffee
- Any applicable policies?
- Security contact(s), teams, ...
- Do some documentation, note times
  - Save outputs
- Do communication
- Isolate the system

# Live analysis

- Part of triage
- Checking live system is often important
  - Access to memory and working system
- Memory can be gathered
  - Hard to hide something
  - Processes are “unlocked”
  - Data can only be available from memory
  - Independent view on OS structures
- But the system may not be yours anymore!

# Performing live analysis

- Before you start, secure evidence that could be changed
  - Snapshot(s), take FS metadata, (RAM)
- Start with introspection of the whole system
  - Network connections, running processes, ....
  - Note processes for additional analysis
- After that, examine suspicious processes
  - resources used
  - recover files
  - obtain memory dumps

# Processes

- A *process* is an instance of a *program*
  - Program is usually an executable file on a disk
- Process keeps data in memory, uses system resources
  - Sometimes released only during termination
- Processes form a hierarchy

# System examination

- Closer look at processes
  - Strange names, executables
  - Distributions of PIDs, relationships, CPU consumption
- Resources in use
  - Memory, open sockets (files, networks), shared memory
- Investigations
  - User-space commands (common commands)
  - Check kernel structures
    - Correlation of command outputs, access lower-level info



# Commands needed

- **Commands**
  - `ps, netstat, lsof`
- **Kernel structures**
  - `/proc/$PID`
- **Document/record the process**
  - Keep track of issued commands
  - Save outputs
    - Ramdisks (`/dev/shm`) might be an option

# /proc records

## /proc/31418

```
-r--r--r-- 1 kouril kouril 0 May 5 18:46 cmdline
lrwxrwxrwx 1 kouril kouril 0 May 5 18:46 cwd -> /tmp
-r----- 1 kouril kouril 0 May 5 18:46 environ
lrwxrwxrwx 1 kouril kouril 0 May 5 18:46 exe -> /usr/bin/wget

dr-x----- 2 kouril kouril 0 May 5 18:46 fd
lrwx----- 1 kouril kouril 64 May 5 18:46 0 -> /dev/pts/47
lrwx----- 1 kouril kouril 64 May 5 18:46 1 -> /dev/pts/47
lrwx----- 1 kouril kouril 64 May 5 18:46 2 -> /dev/pts/47
lrwx----- 1 kouril kouril 64 May 5 18:46 3 -> socket:[3097580]
l-wx----- 1 kouril kouril 64 May 5 18:46 4 -> /tmp/ubuntu-19.04-desktop-
amd64.iso?_ga=2.213675796.1604966281.1557074696-1247976767.1557074696
```

# Deleted files

- Unix keeps deleted files open until they are closed
- `ls /proc/$PID/exe:`
  - `lrwxrwxrwx 1 kouril kouril 0 May 4 07:31 exe -> /tmp/wget (deleted)`
- Proc's "symbolic links" can be used for easy recovering the data
  - `cp/cat/... /proc/$PID/exe /tmp/dest`
  - The process must be still running!
- Both executable and open files (see the `fd` directory)

# Open files

## (lsof -p 31418 -n)

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
wget	31418	kouril	cwd	DIR	252,0	36864	524291	/tmp
wget	31418	kouril	rtd	DIR	252,0	4096	2	/
wget	31418	kouril	txt	REG	252,0	407696	393524	/usr/bin/wget
wget	31418	kouril	mem	REG	252,0	43616	1049154	/lib/x86_64-linux-gnu/ libnss_files-2.19.so
wget	31418	kouril	mem	REG	252,0	3165552	394658	/usr/lib/locale/locale-arc
wget	31418	kouril	mem	REG	252,0	14664	1064472	/lib/x86_64-linux-gnu/libc
wget	31418	kouril	mem	REG	252,0	1857312	1064478	/lib/x86_64-linux-gnu/libc
wget	31418	kouril	mem	REG	252,0	18936	1050745	/lib/x86_64-linux-gnu/libu
wget	31418	kouril	mem	REG	252,0	207128	397935	/usr/lib/x86_64-linux-gnu/
wget	31418	kouril	mem	REG	252,0	100728	1048634	/lib/x86_64-linux-gnu/libz
wget	31418	kouril	mem	REG	252,0	1938752	1050644	/lib/x86_64-linux-gnu/libc
wget	31418	kouril	mem	REG	252,0	387272	1050636	/lib/x86_64-linux-gnu/libS
wget	31418	kouril	mem	REG	252,0	149120	1064460	/lib/x86_64-linux-gnu/ld-2
wget	31418	kouril	mem	REG	252,0	26258	671480	/usr/lib/x86_64-linux-gnu/
wget	31418	kouril	0u	CHR	136,47	0t0	50	/dev/pts/47
wget	31418	kouril	1u	CHR	136,47	0t0	50	/dev/pts/47
wget	31418	kouril	2u	CHR	136,47	0t0	50	/dev/pts/47
wget	31418	kouril	3u	IPv4	3101285	0t0	TCP	127.0.0.1:44280->127.0.0.1
wget	31418	kouril	4w	REG	252,0	14777874	573900	/tmp/ubuntu-19.04-desktop-

# Open network connections (netstat -tnp)

Active Internet connections (w/o servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	127.0.0.1:9050	127.0.0.1:34902	ESTABLISHED	-
tcp	0	0	127.0.0.1:44280	127.0.0.1:8118	ESTABLISHED	31418/wget

# Dumping process memory

- `gcore -p PID -o dump`
  - Part of the GDB package
- Outputs an ELF file (see later) containing the process memory

# Executable file analysis

- Static analysis
- Dynamic analysis

# ELF

## ELF<sup>01</sup>a Linux executable walkthrough

Ange Albertini  
corkami.com



### Dissected file

```
~$ uname -m
x86_64
~$ ./simple64.elf
Hello World!
```

#### Header

technical details for identification and execution

#### simple64.elf sections

contents of the executable

#### header

technical details for linking (ignored for execution)

#### ELF header

ELF header

#### Program Header table

Execution information

#### Code

executable information

#### Data

Information used by the code

#### Sections' names

Sections' names

#### Section Header table

Linking (connecting program objects) information

Hexadecimal dump

ASCII dump

Fields	Values	Explanation
<b>1</b> e_ident	0x7F "ELF"	constant signature
EL_MAG	64 bits, Little-Endian	64 bits, Little-Endian
EL_CLASS, EL_DATA	Always 1	Always 1
EL_VERSION	1	Always 1
e_type	2	Executable
e_machine	0x3E	AMD 64 (and later)
e_version	1	Always 1
e_entry	0x10000080	Address where execution starts
e_phoff	0x40	Program Headers' offset
e_shoff	0xF0	Section Headers' offset
e_ehsize	0x40	ELF header's size
e_phentsize	0x38	Size of a single Program Header
e_phnum	1	Count of Program Headers
e_shentsize	0x40	Size of a single Section Header
e_shnum	4	Count of Section Headers
e_shstrndx	3	Index of the names' section in the table
<b>2</b> p_type	1	The segment should be loaded in memory
p_flags	0	Readable and executable
p_offset	0	Offset where it should be read
p_vaddr	0x10000000	Virtual address where it should be loaded
p_paddr	0x10000000	Physical address where it should be loaded
p_filesz	0xD0	Size on file
p_memsz	0xD0	Size in memory

x64 assembly

Equivalent C code

```
mov rdi, 0x0
mov rsi, 0x100000C0
mov rax, 1
syscall
mov rdi, 1
mov rax, 0x3C
syscall
```

writes(STDOUT\_FILENO, "Hello World\n", len("Hello World\n"));  
 \_exit(1);

Strings

"Hello World\n", 0

Section names

..shstrtab .text .rodata

#### Section Header table

Index	Name	TYPE	FLAGS	ADDRESS	OFFSET	SIZE
0	..crtable	0				
1	.text	1	6	0x10000000	0x40	0x31
2	.rodata	1	2	0x100000C0	0xC0	0xD0
3	.shstrtab	3		0xD0	0x19	

This is the whole file, however, most ELF files contain many more elements. Explanations are simplified, for conciseness.



# Look inside an ELF executable



<https://binvis.io/>

# Static analysis of binary files

- Determine the type

```
file /bin/bash
```

```
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared  
libs), for GNU/Linux 2.6.24, BuildID[sha1]=7e4c4de7a4d259aeb0896fd579609bb6c27fae8d, stripped
```

- Content analysis

- Break down individual ELF sections and analyse them

- .rodata - constants (strings)
- .data - global tables, variables
- .text
- readelf

- Or do quick examination of the file

- Human readable strings
  - `strings -a <binary>`
- Strings often point to username, file paths, function names, . . .
- Malware producers tend to obfuscate important strings
  - XOR, base64, . . .
  - Dynamic calls to library functions
  - `dlopen()`, `dlsym()`

# Countermeasures

- Encoded (packed) binaries
  - Binary is encoded by a customized algorithm and gets unpacked only during executions
  - UPX (easy to unpack), or its modifications
  - Also for scripts – self-executable archives
- Obfuscated scripts
  - very often used for PHP or Javascript

# Next Session

- A joint walk-through the SSC malware
- 35 VMs available for hands-on exercise
- SSH client necessary, access credentials will be circulated