DIGITAL FORENSIC RESEARCH WORKSHOP

National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

DFRWS APAC 2024

Understanding and Analysing ELF

Parag Rughani, Ph. D.
parag.rughani@gmail.com

About me

- Parag H. Rughani, Ph. D.
- Professor of Digital Forensics at National Forensic Sciences University (NFSU), Gandhinagar, Gujarat, India
- DFRWS OC Member
- IEEE Senior Member
- Teach memory forensics, malware analysis and artificial intelligence to Post Graduate students at NFSU.

About me

National Forensic Sciences Uni

National Forensic Sciences University

રાષ્ટ્રીય ફોરેન્સિક વિજ્ઞાન યુનિવર્સિટી

4.7 ★★★★★ (931)

University ·

Overview
Reviews
About
Directions
Save
Nearby
Send to phone
Share
Afghanistan
Islamabad
JAMMU AND KASHMIR
Lahore
HIMACHAL PRADESH
Pakistan
PUNJAB
UTTARAKHAND
HARYANA
Nepal
UTTAR PRADESH
Jaipur
RAJASTHAN
Lucknow
Guwahati
NAGALAND
BIHAR
Karachi
Muscat
National Forensic Sciences University
JHARKHAND
WEST BENGAL
CHHATTISGARH
Surat
Nagpur
ODISHA
MAHARASHTRA
Pune
TELANGANA
GOA
KARNATAKA
ANDHRA PRADESH
Bengaluru
Naypyidaw
Yangon
Bay of Bengal
Arabian Sea
Andaman Sea
Layers
Google

6M56+XP8, Police Bhavan Rd, Sector 9, Gandhinagar, Gujarat 382007

Map data ©2024 Google, Mapa GISrael India Terms Privacy Send Product Feedback 500 km

Introduction

- <u>ELF</u> (Executable and Linking Format) is the main executable file format used on Linux systems.
- You can compare it with <u>Portable Executable (PE)</u> File on Windows.
- User applications, shared libraries, kernel modules, and the kernel itself are all stored in the ELF format.
- Can you name any ELF file that you have used?

The *file* command

- The ***file*** command can be used to check the file type:
  - E.g.: $ *file /bin/ls*
- Which of the following are ELF files?
  - cd
  - cp
  - pwd
  - mv
  - sudo

The *type* command

- Try ***type*** command with the non-elf items from the above list.
- E.g.: *$ type cd*

Output of the *file* command

- Let's understand the output:
  - ELF 64-bit LSB pie executable (Position Independent Executable)
  - x86-64
  - version 1 (SYSV)
  - dynamically linked
  - interpreter /lib64/ld-linux-x86-64.so.2
  - BuildID[sha1]=****************
  - for GNU/Linux 3.2.0
  - **stripped / not stripped**

The ELF File

- Let’s create our first ELF file.
- Prerequisites:
  - Text editor like sublime (you can also use nano or vi if you are comfortable with)
  - gcc (sudo apt install gcc)
- $ *gcc first.c -o first.o*
- $ *file first.o*

The Stripped ELF File

- $ *nm first.o*
- $ *strip first.o -o s_first.o*
- $ *nm s_first.o*

The ELF Header

- The ELF header is located at the very beginning (offset 0) of a file.
- It is represented by an Elf32_Ehdr or Elf64_Ehdr data structure for 32-bit or 64-bit files, respectively.
- The following structure members are important for analysis: e_ident: Holds the file identification information. The first four bytes are "\x7fELF", the fifth byte stores whether the file is 32- or 64-bit, and the sixth byte stores whether the file is in little or big endian format. One can use this signature to scan through memory dumps and find the beginning of ELF files

The ELF Header

- The ELF header is located at the very beginning (offset 0) of a file.
- It is represented by an Elf32_Ehdr or Elf64_Ehdr data structure for 32-bit or 64-bit files, respectively.
- The following structure members are important for analysis:
  - **e_ident:** Holds the file identification information. The first four bytes are “\x7fELF”, the <u>fifth</u> byte stores whether the file is 32-bit or 64-bit, and the <u>sixth</u> byte stores whether the file is in little or big endian format.

The ELF Header

- …
  - **e_type:** Stores the file type, whether it is an executable, relocatable image, shared library, or a core dump.
  - **e_entry:** Holds the program entry point, which is the address of the first instruction that executes when the program is run.
  - **e_phoff, e_phentsize, and e_phnum:** Hold the file offset, entry size, and number of program header entries.

The ELF Header

- …
  - **e_shoff, e_shentsize, and e_shnum:** Hold the file offset, entry size, and number of section header entries.
  - **e_shstrndx:** Stores the index within the section header table of the strings that map to section names.

The ELF Header (Demonstration)

- Open any elf file in a hex viewer (you may like make a copy of the file before opening it.)
- You may like to use ghex (*$ sudo apt install ghex*)
- *$ ghex first.o*
- Observe the first SIX bytes
- Check the output of *$ghex /bin/ls*

readelf

- The -h parameter to *readelf* displays the header information.
- *Examples:*
  - *$ readelf -h first.o*
  - *$ readelf -h /bin/ls*
  - *$ readelf -h s_first.o*

ELF Sections

- An ELF binary is typically divided into multiple sections.
- The **e_shoff** member of the ELF header where the section header entries begin.
- At this offset is an array of Elf32_Shdr or Elf64_Shdr structures that represent each section within the file.
- *$ readelf -S first.o*
- *$ readelf -S s_first.o*

ELF Sections

- The critical members::
  - **sh_name:** Holds an index into the string table of section names.
  - **sh_addr:** Stores the virtual address of where the section will be mapped.
  - **sh_offset:** Holds the offset within the file.
  - **sh_size:** Holds the size of the section in bytes.

Common ELF Sections

- .text: Contains the application’s executable code
- .data: Contains the read/write data (variables)
- .rodata: Contains read-only data
- .plt: Procedure Linkage Table
- .bss: Contains variables that are initialized to zero
- .got: Contains the global offset table

Common ELF Section Types

- **PROGBITS:** Sections whose contents from disk will be loaded into memory upon execution.
- **NOBITS:** Sections that do not have data in the file, but have regions allocated in memory. The .bss is typically a NOBITS section because all its memory is initialized to zero upon execution.
- **STRTAB:** Holds a string table of the application.
- **DYNAMIC:** Indicates that this is a dynamically linked application and holds the dynamic information.
- **HASH:** Contains the hash table of the application's symbols.

Packed ELFs

- Since the section headers are optional, one of the first items removed from an executable by a packer is the section header table.
- This makes reverse engineering difficult because it eliminates the information that analysis tools rely on to statically build a map of the executable.

Exercise 1

- Task A
  - Compare a stripped file (s_first.o) and a regular file (first.o).
  - Do you see same number of sections?
  - If not, which sections are missing / additional?
- Task B
  - Compare two regular files (e.g. first.o and /bin/ls)
  - Do you see same number of sections?
  - If not, which sections are missing / additional?

ELF Symbols

- Symbols are a symbolic reference to some type of data or code such as a global variable or a function.
- .dynsym contains global symbols that refers to the symbols from an external source.
- .symtab contains all the symbols of .dynsm and as well as the local symbols for the executable. E.g. global variables or local functions defined in the code.
- Why 02 symbol tables?
- Check the ‘A’ flag in *$ readelf -S first.o*

ELF Symbols

- Let’s create our second ELF file
- *$ gcc -nostdlib second.c -o second*
- *$ readelf -s first.o | grep main*
- *$ readelf -s s_first.o | grep main*
- *$ readelf -s second | grep func1*
- *$ readelf -s second | grep func2*

Introducing Objdump

- Objdump displays information about one or more object files.
- *$ objdump -d second*
- *$ objdump -d first.o*
- *$ objdump -d s_first.o*
- *$ objdump -d /bin/ls*

ELF Program Headers

- The e_phoff member of the ELF header tells where the program header entries begin.
- At this offset is an Elf32_Phdr or Elf64_Phdr structure.
- The operating system uses program headers to map the file and its sections into memory at runtime.
- When encountering malicious executables, the program header is often the only information available to the analyst to statically analyze the binary.

ELF Program Headers

- *$ readelf -h first.o*
- *$ readelf -h s_first.o*
- *$readelf -h /bin/ls*
- *$ readelf -l first.o*
- *$ readelf -l s_first.o*
- *$ readelf -l bin/ls*

ELF Program Headers

- The important members of this structure are:
  - **p_type:** Describes the type of the segment. Segments are portions of the file that load into memory, and they contain one or more sections of the file. The most common types are PT_LOAD, which describes a segment that must load into memory; PT_DYNAMIC, which describes the dynamic linking information; and PT_INTERP, which holds the full path to the program interpreter.
  - **p_vaddr and p_offset:** Serve the same purpose as sh_addr and sh_offset within the section headers and have the same semantics.

ELF Program Headers

- …
  - **p_filesz:** Holds the size of the segment on disk.
  - **p_memsz:** Holds the size of the segment in memory.

ELF Relocations

- The relocation relies on symbols and sections.
- In relocations, there are **relocation records**, which essentially contain information about how to patch the code related to a given symbol.
- *$ readelf -r second*
- *$ readelf -r s_first.o*

Shared Libraries

- Shared libraries are reusable pieces of code that can be dynamically loaded into an application.
- These files are generally stored on disk with the .so (shared object) extension and can be thought of as counterparts to DLL files on Windows.
- *$ file/lib64/ld-linux-x86-64.so.2*
- *$ readelf -h /lib64/ld-linux-x86-64.so.2*

Shared Libraries

- Due to the power of shared libraries, attackers and malware will often inject shared libraries into processes to steal data, escalate privileges, or maintain persistence.
- ELF files specify which shared libraries they need within the dynamic information section, and this can be read using the -d parameter of readelf and filtering on the “NEEDED” entries.
- *$ readelf -d first.o | grep NEEDED*
- *$ readelf -d second | grep NEEDED*
- *$ readelf -d /bin/ls | grep NEEDED*

ELF Relocations

- Alternatively one can use *objdump*.
- *$ objdump -p first.o | grep NEEDED*
- *$ objdump -p second | grep NEEDED*
- *$ objdump -p /bin/ls | grep NEEDED*

Exercise 2

Analyze sample1 (https://github.com/paragrughani/DFRWS_APAC24) and answer the following questions:

1. Is it a 64 bit or 32 bit ELF?
2. Is it in Little of Big Endian?
3. Is it a stripped file?
4. How many sections are there?
5. Which are the needed .so files?