Linux Incident Response - introducing objdump.

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-11-01

At some point, in any incident response engagement, you are going to have to try and discover how an executable file works. On Linux platforms**,** you have the advantage of being able to use the objdump command.

Structure of a Linux ELF File:

In Linux, executable and object files are typically in the ELF (Executable and Linkable Format) format. ELF files have a well-defined structure, comprising sections, headers, and program segments. The ELF header contains essential information about the file, such as the type of ELF file, architecture, entry point, and section header table's offset.

Different Sections in a Linux ELF File:

ELF files consist of various sections, each serving a specific purpose. Common sections include (executable code), (code executed on load), (initialized data), (uninitialized data), (read-only data), and (symbol table). Each section holds distinct types of information, allowing developers and analysts to understand the file's contents and behaviour.

Objdump Command Line Arguments in General:

The command is a powerful tool for examining ELF files. Its general syntax is:

```
```

Options can vary widely, allowing users to specify the format and level of detail in the output. Objdump can analyze object files, executable files, shared libraries, and even core dumps, providing critical insights into their structures and contents.

The commonly used -d, -x, and -D Arguments:

- : Disassembles the executable sections of the file, displaying the assembly code. This is particularly useful for understanding the file's logic and operations.

```
```

```
$ objdump -d UjDHuoa

UjDHuoa:     file format elf64-x86-64


Disassembly of section .init:

00000000004000e8 <.init>:
  4000e8:       50                      push   %rax
  4000e9:       58                      pop    %rax
  4000ea:       c3                      ret

Disassembly of section .text:

00000000004000f0 <.text>:
  4000f0:       51                      push   %rcx
  4000f1:       31 d2                   xor    %edx,%edx
  4000f3:       31 f6                   xor    %esi,%esi
  4000f5:       41 b8 38 00 00 00       mov    $0x38,%r8d
  4000fb:       b9 01 00 00 00          mov    $0x1,%ecx
  400100:       e8 03 e9 04 00          call   0x44ea08
  400105:       48 85 c0                test   %rax,%rax
  400108:       74 14                   je     0x40011e
  40010a:       c7 00 ff ff ff ff       movl   $0xffffffff,(%rax)
  400110:       c7 40 04 ff ff ff ff    movl   $0xffffffff,0x4(%rax)
  400117:       c7 40 08 ff ff ff ff    movl   $0xffffffff,0x8(%rax)
  40011e:       5a                      pop    %rdx
```

Example output from objdump -d

- : Displays all available header information, including the ELF header, section headers, and program headers. This option provides a detailed overview of the file's structure.

```
$ objdump -x UjDHuoa

UjDHuoa:     file format elf64-x86-64
UjDHuoa
architecture: i386:x86-64, flags 0x00000102:
EXEC_P, D_PAGED
start address 0x00000000004001b5

Program Header:
    LOAD off    0x0000000000000000 vaddr 0x0000000000400000 paddr 0x0000000000400000 align 2**21
         filesz 0x000000000007174c memsz 0x000000000007174c flags r-x
    LOAD off    0x0000000000071750 vaddr 0x0000000000671750 paddr 0x0000000000671750 align 2**21
         filesz 0x00000000000033e0 memsz 0x0000000000005478 flags rw-
   STACK off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**4
         filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .init         00000003  00000000004000e8  00000000004000e8  000000e8  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00058d70  00000000004000f0  00000000004000f0  000000f0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         00000003  0000000000458e60  0000000000458e60  00058e60  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00012270  0000000000458e80  0000000000458e80  00058e80  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .eh_frame     0000665c  000000000046b0f0  000000000046b0f0  0006b0f0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .init_array   00000008  0000000000671750  0000000000671750  00071750  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  6 .fini_array   00000008  0000000000671758  0000000000671758  00071758  2**3
```

Example output from objdump -x

- : Acts as a combination of and , providing both disassembly and all headers' information.

Symbol Tables and Dynamic Symbol Table:

Symbol tables contain information about symbols within the ELF file, including function and variable names. Symbols in this context are the names associated with functions, variables, constants or even sections within the ELF file. The argument in displays the symbol table:

Dynamic symbol tables, displayed using , contain symbols that are dynamically linked, providing insights into function calls and libraries used during runtime:

Using Objdump for Incident Response and Digital Forensics:

During incident response, proves invaluable for analyzing unknown or suspicious executables. By disassembling executable sections (**-d**), analysts can scrutinize the code for malicious behaviour, identifying potential vulnerabilities or backdoor mechanisms. The and options allow detailed examination of file headers and dynamic linking, aiding in understanding the file's dependencies and potential interactions with other system components.

In digital forensics, helps investigators comprehend the internal structure of executables found on compromised systems. Analyzing symbol tables (**-t** and) can reveal function names and library dependencies, shedding light on the file's purpose and potential exploits. By leveraging during analysis, forensic experts gain critical insights into the behaviour and origins of suspicious executables, enabling a more comprehensive investigation.

Objdump's versatility makes it an essential tool for experienced Linux professionals engaged in incident response and digital forensics, providing in-depth visibility into ELF files' intricacies and aiding in the identification of malicious activities within complex executable binaries.

Linux Incident Response Training

You can find out more about this, and more topics related to Linux incident response on the recently released SANS Digital Forensics and Incident Response training course **FOR577 - Linux Incident Response and Threat Hunting.**

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858