

# Linux incident response - the readelf command

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-11-06

## What is readelf?

readelf is a command-line tool that is used to display information about ELF (Executable and Linkable Format) files. It is a part of the GNU Binutils package and is available on most Linux distributions. The readelf command can be used to extract information about the headers, sections, symbols, and other aspects of an ELF file. It is a powerful tool that can be used for general system administration purposes as well as for incident response and malware analysis.

One of the most common uses of the readelf command is to display the headers of an ELF file. The `-h` option can be used to display the ELF file header, which contains information about the file type, machine architecture, entry point address, and other details. For example, the command `readelf -h /usr/bin/ls` will display the ELF file header of the `/usr/bin/ls` binary.

Another useful feature of the readelf command is the ability to display information about the different sections of an ELF file. The `-S` option can be used to display the section headers of the file, which contain information about the various sections such as the code, data, and bss sections. For example, the command `readelf -S /usr/bin/ls` will display the section headers of the `/usr/bin/ls` binary.

The readelf command can also be used to display the symbols table of an ELF file. The `-s` option can be used to display the symbol table, which contains information about the functions and variables defined in the file. For example, the command `readelf -s /usr/bin/ls` will display the symbol table of the `/usr/bin/ls` binary.

In addition to these features, the readelf command can also be used to display information about the dynamic section of an ELF file (`-d` option), the relocation section (`-r` option), and the core notes (`-n` option). The readelf command can also be used to check whether a file is an ELF file (`-a` option) and to display the version of the readelf command (`-v` option).

The readelf command is a powerful tool that can be used for incident response and malware analysis. For example, it can be used to analyze the headers and sections of a binary file to determine whether it is malicious. The readelf command can also be used to extract information about the symbols and functions defined in a binary file, which can be useful for identifying malicious code.

## Commonly used command line arguments

- To display the ELF file header of a binary file:

```
$ readelf -h /usr/bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 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 (Position-Independent Executable file)
  Machine:                  Advanced Micro Devices X86-64
  Version:                  0x1
  Entry point address:      0x6ab0
  Start of program headers: 64 (bytes into file)
  Start of section headers: 136224 (bytes into file)
  Flags:                    0x0
  Size of this header:      64 (bytes)
  Size of program headers:  56 (bytes)
  Number of program headers: 13
  Size of section headers:  64 (bytes)
  Number of section headers: 31
  Section header string table index: 30
```

example readelf -h output using the ls binary

- **Section Headers (-S):** This option displays detailed information about each section in the ELF file. Section headers include names, types, addresses, sizes, and other attributes.

```
$ readelf -S /usr/bin/ls
There are 31 section headers, starting at offset 0x21420:
```

Section Headers:

| [Nr] | Name              | Type     | Address          | Offset   |
|------|-------------------|----------|------------------|----------|
|      | Size              | EntSize  | Flags Link Info  | Align    |
| [ 0] | 0000000000000000  | NULL     | 0000000000000000 | 00000000 |
| [ 1] | .interp           | PROGBITS | 0000000000000318 | 00000318 |
| [ 2] | .note.gnu.pr[...] | NOTE     | 0000000000000338 | 00000338 |
| [ 3] | .note.gnu.bu[...] | NOTE     | 0000000000000368 | 00000368 |
| [ 4] | .note.ABI-tag     | NOTE     | 000000000000038c | 0000038c |
| [ 5] | .gnu.hash         | GNU_HASH | 00000000000003b0 | 000003b0 |
| [ 6] | .dynsym           | DYNSYM   | 0000000000000400 | 00000400 |
| [ 7] | .dynstr           | STRTAB   | 0000000000000f88 | 00000f88 |
| [ 8] | .gnu.version      | VERSYM   | 000000000000152e | 0000152e |
| [ 9] | .gnu.version_r    | VERNEED  | 0000000000001628 | 00001628 |
| [10] | .rela.dyn         | RELA     | 00000000000016e8 | 000016e8 |

Truncated output from readelf -S looking at the /usr/bin/ls binary

- **Symbol Table (-s):** The symbol table contains information about symbols used in the binary, including functions and variables. This is invaluable for understanding program logic.

```
$ readelf -s /usr/bin/ls
```

```
Symbol table '.dynsym' contains 123 entries:
```

| Num: | Value            | Size | Type   | Bind   | Vis     | Ndx | Name                    |
|------|------------------|------|--------|--------|---------|-----|-------------------------|
| 0:   | 0000000000000000 | 0    | NOTYPE | LOCAL  | DEFAULT | UND |                         |
| 1:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.3 (2)   |
| 2:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 3:   | 0000000000000000 | 0    | OBJECT | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 4:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 5:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.3.4 (4) |
| 6:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | raise@GLIBC_2.2.5 (3)   |
| 7:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.34 (5)  |
| 8:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | abort@GLIBC_2.2.5 (3)   |
| 9:   | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 10:  | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 11:  | 0000000000000000 | 0    | NOTYPE | WEAK   | DEFAULT | UND | _ITM_deregisterT[...]   |
| 12:  | 0000000000000000 | 0    | OBJECT | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 13:  | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 14:  | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | _exit@GLIBC_2.2.5 (3)   |
| 15:  | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.2.5 (3) |
| 16:  | 0000000000000000 | 0    | FUNC   | GLOBAL | DEFAULT | UND | __[...]@GLIBC_2.4 (6)   |

Truncated example output from readelf -s looking at /usr/bin/ls

- **Dynamic Section (-d):** This option reveals the dynamic linking information, including shared library dependencies and versioning details.

```
$ readelf -d /usr/bin/ls
```

```
Dynamic section at offset 0x20a98 contains 24 entries:
```

| Tag                 | Type         | Name/Value                        |
|---------------------|--------------|-----------------------------------|
| 0x0000000000000001  | (NEEDED)     | Shared library: [libselinux.so.1] |
| 0x0000000000000001  | (NEEDED)     | Shared library: [libc.so.6]       |
| 0x000000000000000c  | (INIT)       | 0x4000                            |
| 0x000000000000000d  | (FINI)       | 0x17134                           |
| 0x0000000006ffffef5 | (GNU_HASH)   | 0x3b0                             |
| 0x0000000000000005  | (STRTAB)     | 0xf88                             |
| 0x0000000000000006  | (SYMTAB)     | 0x400                             |
| 0x000000000000000a  | (STRSZ)      | 1446 (bytes)                      |
| 0x000000000000000b  | (SYMENT)     | 24 (bytes)                        |
| 0x0000000000000015  | (DEBUG)      | 0x0                               |
| 0x0000000000000003  | (PLTGOT)     | 0x21c58                           |
| 0x0000000000000002  | (PLTRELSZ)   | 2400 (bytes)                      |
| 0x0000000000000014  | (PLTREL)     | RELA                              |
| 0x0000000000000017  | (JMPREL)     | 0x2ac8                            |
| 0x0000000000000007  | (RELA)       | 0x16e8                            |
| 0x0000000000000008  | (RELASZ)     | 5088 (bytes)                      |
| 0x0000000000000009  | (RELAENT)    | 24 (bytes)                        |
| 0x000000000000001e  | (FLAGS)      | BIND_NOW                          |
| 0x0000000006fffffb  | (FLAGS_1)    | Flags: NOW PIE                    |
| 0x0000000006fffffe  | (VERNEED)    | 0x1628                            |
| 0x0000000006ffffff  | (VERNEEDNUM) | 2                                 |
| 0x0000000006fffff0  | (VERSYM)     | 0x152e                            |
| 0x0000000006fffff9  | (RELACOUNT)  | 199                               |
| 0x0000000000000000  | (NULL)       | 0x0                               |

Output of readelf -d examining /usr/bin/ls

- **Program Headers (-l):** Program headers describe segments' layout in the binary. This information is vital for understanding memory mapping and the executable's load address.

```
$ readelf -l /usr/bin/ls

Elf file type is DYN (Position-Independent Executable file)
Entry point 0x6ab0
There are 13 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
               FileSiz          MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
               0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
               0x000000000000001c 0x000000000000001c  R      0x1
    [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
               0x00000000000003428 0x00000000000003428  R      0x1000
  LOAD           0x0000000000000400 0x0000000000000400 0x0000000000000400
               0x000000000000013146 0x000000000000013146  R E    0x1000
  LOAD           0x000000000000018000 0x000000000000018000 0x000000000000018000
               0x00000000000007458 0x00000000000007458  R      0x1000
  LOAD           0x000000000000020000 0x000000000000021000 0x000000000000021000
               0x00000000000001278 0x00000000000002540  RW     0x1000
  DYNAMIC         0x000000000000020a98 0x000000000000021a98 0x000000000000021a98
               0x000000000000001c0 0x000000000000001c0  RW     0x8
  NOTE           0x00000000000000338 0x00000000000000338 0x00000000000000338
               0x00000000000000030 0x00000000000000030  R      0x8
  NOTE           0x00000000000000368 0x00000000000000368 0x00000000000000368
               0x00000000000000044 0x00000000000000044  R      0x4
```

```
GNU_PROPERTY 0x00000000000000338 0x00000000000000338 0x00000000000000338
0x00000000000000030 0x00000000000000030 R 0x8
GNU_EH_FRAME 0x00000000000001cdcc 0x00000000000001cdcc 0x00000000000001cdcc
0x0000000000000056c 0x0000000000000056c R 0x4
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x00000000000000000 0x0000000000000000 RW 0x10
GNU_RELRO 0x00000000000002000 0x000000000000021000 0x000000000000021000
0x00000000000001000 0x00000000000001000 R 0x1

Section to Segment mapping:
Segment Sections...
00
01 .interp
02 .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .re
03 .init .plt .plt.got .plt.sec .text .fini
04 .rodata .eh_frame_hdr .eh_frame
05 .ctors .dtors .data.rel.ro .dynamic .got .data .bss
06 .dynamic
07 .note.gnu.property
08 .note.gnu.build-id .note.ABI-tag
09 .note.gnu.property
10 .eh_frame_hdr
11
12 .ctors .dtors .data.rel.ro .dynamic .got
```

Two screenshots of output from readelf -l

- **Section Headers and Their Contents (-x):** This option displays the hexadecimal and ASCII representation of the content of a specific section. For instance, to examine the .text section:

```
$ readelf -x .text /usr/bin/ls
```

```
Hex dump of section '.text':
```

```
0x00004cf0 50e80afa fffffe805 faffffe8 00faffff P.....
0x00004d00 e8fbf9ff ffe8f6f9 fffffe8f1 f9ffffe8 .....
0x00004d10 ecf9ffff e8e7f9ff ffe8e2f9 ffff6690 .....f.
0x00004d20 f30f1efa 41574156 41554154 55534883 ....AWAVAUATUSH.
0x00004d30 ec78488b 2e64488b 04252800 00004889 .xH..dH..%(...H.
0x00004d40 44246831 c04885ed 0f84a61c 00004189 D$h1.H.....A.
0x00004d50 fe4889f3 4889efbe 2f000000 e8affbfff .H..H.../.....
0x00004d60 ff4989c4 4885c074 4f4c8d68 014c89e8 .I..H..t0L.h.L..
0x00004d70 4829e848 83f8067e 3f498d7c 24faba07 H).H...~?I.|$...
0x00004d80 00000048 8d35e34e 0100e891 f9ffff85 ...H.5.N.....
0x00004d90 c07525ba 03000000 488d35d6 4e01004c .u%.....H.5.N..L
0x00004da0 89ef4c89 ede876f9 ffff85c0 0f84d611 ..L...v.....
0x00004db0 0000660f 1f440000 488b0509 d2010048 ..f..D..H.....H
0x00004dc0 892d92d6 0100bf06 00000048 8d35eb4d .-.....H.5.M
0x00004dd0 01004889 28488d2d ca4b0100 e84ffdfdf ..H.(H.-.K...0..
0x00004de0 ff488d35 914e0100 4889efe8 60faffff .H.5.N..H...`...
0x00004df0 4889efe8 18faffff 488d3de1 6c0000c7 H.....H.=.l...
0x00004e00 0577d201 00020000 00e8c222 0100bf01 .w....."....
0x00004e10 00000048 b8000000 00000000 80c70545 ...H.....E
0x00004e20 d9010000 000000c6 053ad901 000148c7 .....:....H.
0x00004e30 0527d901 00000000 00488905 10d90100 .'.....H.....
```

Example showing the .text section of /usr/bin/ls

```

$ readelf -x .text /cases/binaries/freeciv

Hex dump of section '.text':
 0x0804abd0 8d4c2404 83e4f0ff 71fc5589 e55731ff .L$.....q.U..Wl.
 0x0804abe0 56535183 ec68c745 b4fc4a07 08e8ceff VSQ..h.E..J.....
 0x0804abf0 ffff85c0 0f85f603 00008d45 e1575068 .....E.WPh
 0x0804ac00 10160608 8d45b850 e8b3faff ff83c40c .....E.P.....
 0x0804ac10 8d45e250 68151606 088d45bc 50e89efa .E.Ph.....E.P...
 0x0804ac20 ffff83c4 0c8d45e3 50681e16 06088d45 .....E.Ph.....E
 0x0804ac30 c050e889 faffffc7 04241800 0000e84d .P.....$. ....M
 0x0804ac40 fbffff83 c41089c7 8d5dc053 8d45bc50 .....].S.E.P
 0x0804ac50 8d45b850 57e8dc0f 0000891c 24e8eef8 .E.PW.....$. ...
 0x0804ac60 ffff8d45 bc890424 e8e3f8ff ff8d45b8 ...E...$. ....E.
 0x0804ac70 890424e8 d8f8ffff 893c24e8 9a0c0000 ..$. ....<$. ....
 0x0804ac80 83c41085 c00f84b2 03000083 ec0c57e8 .....W.
 0x0804ac90 94260000 83c41084 c00f859e 03000083 .&.....
 0x0804aca0 ec0c57e8 b2270000 83c41084 c0752d83 ..W..'.....u-.
 0x0804acb0 ec0c57e8 76270000 83c41084 c00f842d ..W.v'.....-
 0x0804acc0 03000083 ec0c57e8 70120000 83c41084 .....W.p.....
 0x0804acd0 c00f850e 030000e9 14030000 56568d5d .....VV.]
 0x0804ace0 c45753e8 c00d0000 8d45b489 1c2450e8 .WS.....E...$P.
 0x0804acf0 0cfcffff 891c2431 f6e852f8 ffff83c4 .....$1..R.....
 0x0804ad00 10e9c400 000083c4 0c8d45e6 50681e16 .....E.Ph..
 0x0804ad10 06088d45 d050e8a5 f9ffffc7 04240c00 ...E.P.....$. ..
 0x0804ad20 0000e869 faffff83 c41089c6 8d5dd053 ...i.....].S
 0x0804ad30 8d45cc50 8d45c850 56e8662e 0000891c .E.P.E.PV.f.....
 0x0804ad40 24e80af8 ffff8d45 cc890424 e8fff7ff $. ....E...$. ....
 0x0804ad50 fff8d45c 890424e8 f4f7ffff 892424e8 .E...$. ....4$

```

Example showing the .text section of a malware sample (fysbis)

- **Display Dynamic String Table (--dyn-syms):** This option prints dynamic symbols, which are essential for dynamic linking analysis.

```
$ readelf --dyn-syms /usr/bin/ls

Symbol table '.dynsym' contains 123 entries:
  Num:      Value              Size Type      Bind   Vis      Ndx Name
   0: 0000000000000000          0 NOTYPE   LOCAL  DEFAULT UND
   1: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND __[...]@GLIBC_2.3 (2)
   2: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
   3: 0000000000000000          0 OBJECT   GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
   4: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
   5: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.3.4 (4)
   6: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND raise@GLIBC_2.2.5 (3)
   7: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.34 (5)
   8: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND abort@GLIBC_2.2.5 (3)
   9: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
  10: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
  11: 0000000000000000          0 NOTYPE   WEAK    DEFAULT UND _ITM_deregisterT[...]
  12: 0000000000000000          0 OBJECT   GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
  13: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
  14: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND _exit@GLIBC_2.2.5 (3)
  15: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.2.5 (3)
  16: 0000000000000000          0 FUNC    GLOBAL  DEFAULT UND [...]@GLIBC_2.4 (6)
```

Truncated output from readelf --dyn-syms run against the /usr/bin/ls binary

Understanding the output of the readelf command empowers professionals to dissect ELF binaries systematically, making it an indispensable tool for incident responders and security analysts dealing with Linux-based systems and potential security threats.

#linux #elf #binaryfiles #fileanalysis #forensics #incidentresponse #dfir #cybersecurity

---

## 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](http://www.halkynconsulting.co.uk/security-resources).

**Web:** [www.halkynconsulting.co.uk](http://www.halkynconsulting.co.uk) | **Email:** [info@halkynconsulting.co.uk](mailto:info@halkynconsulting.co.uk) | **Tel:** +44 1244 940858