Fundamentals

Introduction to Binary Files

Yan Shoshitaishvili
Arizona State University

`/bin/cat`

What is cat?

```
yans@cse466 ~ $ file /bin/cat
/bin/cat: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=747e524bc20d33ce25ed4aea108e3025e5c3b78f, stripped
```

What is cat?

```
yans@cse466 ~ $ file /bin/cat
/bin/cat: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=747e524bc20d33ce25ed4aea108e3025e5c3b78f, stripped
```

What is cat?

```
yans@cse466 ~ $ file /bin/cat
/bin/cat: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=747e524bc20d33ce25ed4aea108e3025e5c3b78f, stripped
```

What is an ELF?

ELF is a binary file format.

Contains the program and its data.
Describes how the program should be loaded (*program/segment headers*).
Contains metadata describing program components (*section headers*).

ELF Program Headers

Program headers specify information needed to prepare the program for execution. Most important entry types:

INTERP: defines the library that should be used to load this ELF into memory.
LOAD: defines a part of the file that should be loaded into memory.

Program headers are *the* source of information used when loading a file.

ELF Section Headers

A different view of the ELF with useful information for introspection, debugging, etc.

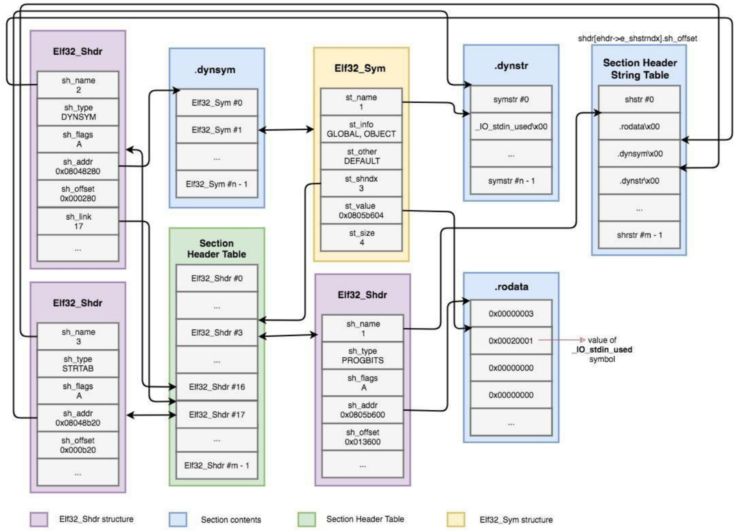
Important sections:

.text: the executable code of your program.
.plt and **.got:** used to resolve and dispatch library calls.
.data: used for pre-initialized global writable data (such as global arrays with initial values)
.rodata: used for global read-only data (such as string constants)
.bss: used for uninitialized global writable data (such as global arrays without initial values)

Section headers are *not* a necessary part of the ELF: only segments (defined via program headers) are needed for loading and operation! Section headers are just metadata.

Symbols

Binaries (and libraries) that use dynamically loaded libraries rely *symbols* (names) to find libraries, resolve function calls into those libraries, etc.



Source (and further reading): https://www.intezer.com/blog/elf/executable-linkable-format-101-part-2-symbols/

Interacting with your ELF

Several ways to dig in:

gcc to make your ELF.
readelf to parse the ELF header.
objdump to parse the ELF header and disassemble the source code.
nm to view your ELF's symbols.
patchelf to change some ELF properties.
objcopy to swap out ELF sections.
strip to remove otherwise-helpful information (such as symbols).
kaitai struct (https://ide.kaitai.io/) to look through your ELF interactively.

Further reading:
https://www.intezer.com/blog/research/executable-linkable-format-101-part1-sections-segments/
https://www.intezer.com/blog/elf/executable-linkable-format-101-part-2-symbols/
https://www.intezer.com/blog/elf/executable-and-linkable-format-101-part-3-relocations/
https://www.intezer.com/blog/elf/executable-linkable-format-101-part-4-dynamic-linking/