PWN College

Session 3

Atousa Ahsani

Main Reference: https://pwn.college/

Fundamentals

Computer Architecture

Assembly Code

Introduction to Binary Files

Linux Process Loading

Linux Process Execution

What is an ELF?

- Contains the **program** and its **data**.
  - **Program/Segment Headers**: Describes how the program should be **loaded**.
  - **Section Headers**: Contains **metadata** describing program **components**.

Symbols

- **Symbols** are references to certain types of **data** or **codes** by the program. Typical symbols include **global variables**, **function names**, etc.
- Symbols are **linker** and **debugger** information necessary.
- Two symbol tables:
  - .**symtab**: It contains all symbols of inside ELF file.
  - .**dynsym**: It is a subset of .symtab which contains symbols related to linker.

Interacting with your ELF

- **gcc** to make your ELF.
- **readelf** to parse the ELF header.
- **kaitai struct** (https://ide.kaitai.io/) to look through your ELF interactively.

Interacting with your ELF

- **objdump** to parse the ELF header and disassemble the source code.
  - *objdump −h cat*

```
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000000318  0000000000000318  00000318  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.gnu.property 00000020  0000000000000338  0000000000000338  00000338  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000000358  0000000000000358  00000358  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .note.ABI-tag 00000020  000000000000037c  000000000000037c  0000037c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     00000024  00000000000003a0  00000000000003a0  000003a0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .dynsym       00000108  00000000000003c8  00000000000003c8  000003c8  2**3
```

  - *objdump −d cat*

```
Disassembly of section .text:

00000000000010e0 <_start>:
    10e0:       f3 0f 1e fa             endbr64
    10e4:       31 ed                   xor    %ebp,%ebp
    10e6:       49 89 d1                mov    %rdx,%r9
    10e9:       5e                      pop    %rsi
    10ea:       48 89 e2                mov    %rsp,%rdx
    10ed:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
    10f1:       50                      push   %rax
    10f2:       54                      push   %rsp
    10f3:       4c 8d 05 26 02 00 00    lea    0x226(%rip),%r8        # 1320 <__libc_csu_fini>
    10fa:       48 8d 0d af 01 00 00    lea    0x1af(%rip),%rcx        # 12b0 <__libc_csu_init>
    1101:       48 8d 3d c1 00 00 00    lea    0xc1(%rip),%rdi        # 11c9 <main>
    1108:       ff 15 d2 2e 00 00       callq  *0x2ed2(%rip)        # 3fe0 <__libc_start_main@GLIBC_2.2.5>
    110e:       f4                      hlt
    110f:       90                      nop
```

Interacting with your ELF

- **nm** to view your ELF's symbols.
  - *nm − a cat*

```
0000000000000000 a
0000000000004010 b .bss
0000000000004010 B __bss_start
0000000000000000 a cat.c
0000000000000000 n .comment
0000000000004010 b completed.8060
0000000000000000 a crtstuff.c
0000000000000000 a crtstuff.c
                 w __cxa_finalize@@GLIBC_2.2.5
0000000000004000 d .data
0000000000004000 D __data_start
0000000000004000 W data_start
0000000000001110 t deregister_tm_clones
0000000000001180 t __do_global_dtors_aux
0000000000003da0 d __do_global_dtors_aux_fini_array_entry
0000000000004008 D __dso_handle
0000000000003da8 d .dynamic
0000000000003da8 d _DYNAMIC
00000000000004d0 r .dynstr
00000000000003c8 r .dynsym
0000000000004010 D _edata
0000000000002050 r .eh_frame
000000000000200c r .eh_frame_hdr
0000000000004018 B _end
0000000000001328 t .fini
0000000000001328 T _fini
0000000000003da0 d .fini_array
00000000000011c0 t frame_dummy
0000000000003d98 d __frame_dummy_init_array_entry
0000000000002154 r __FRAME_END__
0000000000003f98 d _GLOBAL_OFFSET_TABLE_
```

  - *nm − D cat*

```
                 w __cxa_finalize
                 w __gmon_start__
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
                 U __libc_start_main
                 U open
                 U puts
                 U read
                 U __stack_chk_fail
                 U write
```

Interacting with your ELF

- objcopy

```
→  2- BinaryFiles objcopy --dump-section .rodata=out.txt ./cat

→  2- BinaryFiles cat out.txt
Hello%
→  2- BinaryFiles subl out.txt        0100 0200 4865 6c6c 6f00 --> 0100 0200 4869 6969 6900
→  2- BinaryFiles objcopy --update-section .rodata=out.txt ./cat

→  2- BinaryFiles ./cat cat.c
Hiiii
int main(int argc, char **argv)
{
        char buf[1024];
        int n;
        int fd = argc == 1 ? 0 : open(argv[1], 0);
        puts("Hello");
        while((n = read(fd, buf, 1024)) > 0 && write(1, buf, n) > 0);
}%
```

Interacting with your ELF

- **strip** to remove otherwise-helpful information (such as symbols). It removes **symbols** and **sections** from files.

```
→  2- BinaryFiles strip cat
→  2- BinaryFiles nm -a cat
nm: cat: no symbols
```

Fundamentals

Computer Architecture

Assembly Code

Introduction to Binary Files

Linux Process Loading

Linux Process Execution

Linux Process Loading

- Cat

```
1  int main(int argc, char **argv)
2  {
3      char buf[1024];
4      int n;
5      int fd = argc == 1 ? 0 : open(argv[1], 0);
6
7      while((n = read(fd, buf, 1024)) > 0 && write(1, buf, n) > 0);
8  }
```

Cat Lifecycle

1. **A process is created.**
2. Cat is loaded.
3. Cat is initialized.
4. Cat is launched.
5. Cat reads its arguments and environment.
6. Cat does its thing.
7. Cat terminates.

A Process is Created

- What is a process?
  - A **process** is simply a running **application**, **command**, or any other **program**.
  - Every individual **program** on your computer runs as **process**.

Process Attributes

- **State** (running, waiting, stopped, zombie)
- **Priority** (and other scheduling information)
- **Parent**, **Children, Siblings**
  - **Parent**: A process that created current process.
  - **Children**: Processes that current process creates.
  - **Sibling**: Other processes created by parent process.
- **Shared Resources** (files, pipes, sockets)
- **Virtual Memory Space**
- **Security Context**
  - Effective uid and gid
  - Saved uid and gid
  - Capabilities

Where do processes come from?

- **fork** and (more recently) **clone** are **system calls** that create a nearly exact **copy** of the calling process.
  - a **parent** and a **child**.
- **fork()** creates a **new process** by duplicating the calling process. The child process and the parent process run in **separate memory spaces**.
- **clone()** create a new child process, in a manner similar to fork(). It provides more precise **control** over what pieces of execution context are **shared** between the **calling** process and the **child** process.

Where do processes come from?

- Later, the **child** process usually uses the ***execve*** syscall to replace itself with another process.

- Example:
  - You type ***/bin/cat*** in bash.
  - Bash **forks** itself into the old **parent** process and the **child** process.
  - The **child** process ***execves*** /bin/cat, becoming /bin/cat.

Cat Lifecycle

1. A process is created.
2. **Cat is loaded.**
3. Cat is initialized.
4. Cat is launched.
5. Cat reads its arguments and environment.
6. Cat does its thing.
7. Cat terminates.

Can we load?

- Before anything is loaded, the kernel checks for **executable permissions**.
- If a file is not executable, ***execve*** will fail.

What to load?

- To figure out what to load, the **Linux kernel** reads the **beginning** of the file (i.e., /bin/cat), and makes a decision:
  1. If the file starts with **#!** (**Shebang**), the kernel extracts the **interpreter** from the rest of that line and executes this interpreter with the **original file** as an **argument**.

```
→  2-Shabeng cat some_script
#!/bin/sh
echo "hiiiii"
→  2-Shabeng ./some_script
zsh: permission denied: ./some_script
→  2-Shabeng chmod +x some_script
→  2-Shabeng ./some_script
hiiiii
```

```
→  2-Shabeng cat some_script
echo "hiiiii"%
→  2-Shabeng /bin/sh some_script
hiiiii
```

What to load? (cont'd)

2. If the file matches a format in ***/proc/sys/fs/binfmt_misc***, the kernel executes the interpreter specified for that **format** with the original file as an **argument**.

```
➜  ~ cd /proc/sys/fs/binfmt_misc
➜  binfmt_misc ls
jar  llvm-10-runtime.binfmt  python2.7  python3.8  register  status
➜  binfmt_misc cat jar
enabled
interpreter /usr/bin/jexec
flags:
offset 0
magic 504b0304
➜  binfmt_misc cat python3.8
enabled
interpreter /usr/bin/python3.8
flags:
offset 0
magic 550d0d0a
```

```
➜  __pycache__ ls
result.cpython-38.pyc
➜  __pycache__ chmod +x result.cpython-38.pyc
➜  __pycache__ ./result.cpython-38.pyc 1
wrong!
```

What to load? (cont'd)

3. If the file is a **dynamically-linked ELF**, the kernel reads the **interpreter/loader** defined in the **ELF**, loads the interpreter and the original file, and lets the interpreter take control.
4. If the file is a **statically-linked ELF**, the **kernel** will load it.
5. Other legacy file formats are checked for.

- These can be **recursive**!

Dynamically linked ELFs: the interpreter

- Process loading is done by the **ELF interpreter** specified in the binary.

```
→  1-LinuxProcessLoading_cat gcc cat.c -o cat_dyn
→  1-LinuxProcessLoading_cat readelf -a cat_dyn | grep interpreter
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

- Interpreter: `/lib64/ld-linux-x86-64.so.2`

Dynamically linked ELFs: the interpreter

- Interpreter can be overridden

```
→  1-LinuxProcessLoading_cat /lib64/ld-linux-x86-64.so.2 ./cat_dyn cat.c
int main(int argc, char **argv)
{
        char buf[1024];
        int n;
        int fd = argc == 1 ? 0 : open(argv[1], 0);
        while((n = read(fd, buf, 1024)) > 0 && write(1, buf, n) > 0);
}
```

- Or changed.

```
→  1-LinuxProcessLoading_cat patchelf --set-interpreter /some/interpreter ./cat_dyn
→  1-LinuxProcessLoading_cat readelf -a cat_dyn| grep interpreter
      [Requesting program interpreter: /some/interpreter]
→  1-LinuxProcessLoading_cat ./cat_dyn
zsh: no such file or directory: ./cat_dyn
```

Dynamically linked ELFs: the loading process

1. The **program** and its **interpreter** are loaded by the **kernel**.
2. The **interpreter** locates the **libraries**.
3. The interpreter loads the libraries.

How does interpreter locate the libraries?

1. All **Preloading Libraries**
   - **Preloading** a **library** means that its **functions** will be used **before others** of the **same name** in later libraries.
   - LD_PRELOAD
     - An **environmental variable** containing one or more **paths** to shared **libraries**, or shared **objects**.
   - */etc/ld.so.preload*
     - File containing a whitespace-separated **list** of **ELF shared objects** to be loaded before the program.
   - If **both** LD_PRELOAD and */etc/ld.so.preload* are employed, the libraries specified by **LD_PRELOAD** are preloaded first.

LD_PRELOAD Example

```
→  3-LinuxProcessLoading_preload cat preload.c
int read(int fd, char *buf, int n){
        buf[0] = 'H';
        buf[1] = 'i';
        buf[2] = 'i';
        buf[3] = 'i';
        buf[4] = '\n';
        return 5;
}
→  3-LinuxProcessLoading_preload gcc -shared preload.c -o preload.so
→  3-LinuxProcessLoading_preload LD_PRELOAD=./preload.so ./cat cat
Hiii
Hiii
Hiii
Hiii
Hiii^C
```

```
→  3-LinuxProcessLoading_preload strace -E LD_PRELOAD=./preload.so ./cat cat | head -n 100
execve("./cat", ["./cat", "cat"], 0x5576d7d174f0 /* 55 vars */) = 0
brk(NULL)                               = 0x563ef6f84000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc9de00bc0) = -1 EINVAL (Invalid argument)
openat(AT_FDCWD, "./preload.so", O_RDONLY|O_CLOEXEC) = 3
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

How does interpreter locate the libraries?

2. **Library Paths**
   - LD_LIBRARY_PATH
     - A colon-separated **set of directories** where libraries should be searched for first, before the **standard** set of directories.
     - This is useful when **debugging** a **new library** or using a **nonstandard library** for special purposes.

```
→  3-LinuxProcessLoading_preload strace -E LD_LIBRARY_PATH=/some/lib/path ./cat cat | head -n 100
execve("./cat", ["./cat", "cat"], 0x55ef8c6b94f0 /* 55 vars */) = 0
brk(NULL)                               = 0x558958714000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffce93ace60) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/haswell/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls/haswell/x86_64", 0x7ffce93ac0b0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/haswell/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls/haswell", 0x7ffce93ac0b0) = -1 ENOENT (No such file or directory)
```

How does interpreter locate the libraries?

3. **Run-time** search **Path**
   - rpath
     - It determines the **run-time search path** hard-coded in an executable file or library.
     - DT_RUNPATH or DT_RPATH
     - **Dynamic Section**
       - Contains **information** used by the **ELF interpreter** to setup the binary.

```
→  3-LinuxProcessLoading_preload patchelf --set-rpath /some/runpath ./cat
→  3-LinuxProcessLoading_preload readelf -d cat

Dynamic section at offset 0x5000 contains 28 entries:
  Tag        Type                         Name/Value
 0x000000000000001d (RUNPATH)            Library runpath: [/some/runpath]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

```
→  3-LinuxProcessLoading_preload strace -E LD_LIBRARY_PATH=/some/lib/path -E LD_PRELOAD=./somepreload.so ./cat cat | head -n 100

execve("./cat", ["./cat", "cat"], 0x5648426df4f0 /* 56 vars */) = 0
brk(NULL)                               = 0x55c4b908b000
arch_prctl(0x3001 /* ARCH ??? */, 0x7ffcf70d8950) = -1 EINVAL (Invalid argument)
openat(AT_FDCWD, "./somepreload.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
writev(2, [{iov_base="ERROR: ld.so: object '", iov_len=22}, {iov_base="./somepreload.so", iov_len=16}, {iov_base="' from ", iov_len
{iov_base=" cannot be preloaded (", iov_len=22}, {iov_base="cannot open shared object file", iov_len=30}, {iov_base="): ignored.\n"
omepreload.so' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
) = 119
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/haswell/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls/haswell/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/haswell/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls/haswell", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/tls/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/tls", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/haswell/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/haswell/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/haswell/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/haswell", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/lib/path/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/lib/path", 0x7ffcf70d7ba0)  = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/tls/haswell/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/tls/haswell/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/tls/haswell/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/tls/haswell", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/tls/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/tls/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/tls/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/tls", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/haswell/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/haswell/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/haswell/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/haswell", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath/x86_64", 0x7ffcf70d7ba0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/some/runpath/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/some/runpath", 0x7ffcf70d7ba0)   = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=111655, ...}) = 0
```

How does interpreter locate the libraries?

4. **System-wide Configuration**
   - **ldconfig** is a linux command that creates the necessary links and cache to the most recent shared libraries found in the directories specified on the command line, in the file */etc/ld.so.conf*, and in the trusted directories, */lib* and */usr/lib*
   - */etc/ld.so.conf*
     - File containing a **list** of **directories**, one per line, in which to search for libraries.

5. ***/lib* and */usr/lib***

- Note
  - */etc/ld.so.cache*: File containing an ordered list of libraries found in the directories specified in */etc/ld.so.conf*, as well as those found in the trusted directories.