CS451 – Software Analysis

Lecture 5

libbfd

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

Binary Format Descriptor (BFD)

- GNU binutils
  - A collection of tools that process binaries
  - Inspect binary files, perform small tasks (disassembly, archiving, modifications, etc.)
- Portable operation
  - GNU software works in many different platforms (operating system, hardware architecture)
  - Need to have a common API for working with binaries across different platforms
  - Use objdump to read a PE in Intel or an ELF in ARM

libbfd

- A library that implements the BFD API
- A common abstraction for different binary objects across different platforms
- Implements complex tasks
  - Parsing the format, dealing with byte order, etc.
- Several other GNU systems use it (e.g., gdb)
- Not the best API

How to see if a program uses libbfd?

```
$ ldd /bin/objdump
linux-vdso.so.1 (0x00007ffdc31ea000)
libopcodes-2.30-108.el8_5.1.so => /lib64/libopcodes-2.30-108.el8_5.1.so
(0x00007fbcc4e71000)
libbfd-2.30-108.el8_5.1.so => /lib64/libbfd-2.30-108.el8_5.1.so (0x00007fbcc4b08000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007fbcc4904000)
libc.so.6 => /lib64/libc.so.6 (0x00007fbcc453f000)
libz.so.1 => /lib64/libz.so.1 (0x00007fbcc4328000)
/lib64/ld-linux-x86-64.so.2 (0x00007fbcc547c000)

$ ldd /bin/readelf
linux-vdso.so.1 (0x00007fffa39fc000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f5326589000)
libc.so.6 => /lib64/libc.so.6 (0x00007f53261c4000)
/lib64/ld-linux-x86-64.so.2 (0x00007f5326a2c000)

$ ldd /bin/file
linux-vdso.so.1 (0x00007fffbd7e5000)
libmagic.so.1 => /lib64/libmagic.so.1 (0x00007f26674ed000)
libz.so.1 => /lib64/libz.so.1 (0x00007f26672d6000)
libc.so.6 => /lib64/libc.so.6 (0x00007f2666f11000)
/lib64/ld-linux-x86-64.so.2 (0x00007f266791a000)
```

A file(1) tool based on libbfd

- The file(1) tool opens files and outputs some information related to them
- Files can be simple text, documents, visual representations, binaries, etc.
- file(1) does not use libbfd, because it is not binary focused
- We will create a minimal file(1) tool, just for binaries, using libbfd

Preliminaries

$ file /bin/ls

/bin/ls: **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]=056dda3f1b77919163a7de5563a2b9d9d245554c, **stripped**

Requirements

```
$ yum list installed | grep binutils
binutils.x86_64              2.30-108.el8_5.1 @baseos
binutils-devel.x86_64        2.30-108.el8_5.1 @appstream
```

- Useful file
  - /usr/include/bfd.h

Compilation and run

```
$ gcc -Wall min_file.c -lbfd -o min_file

$ ./min_file /bin/ls
filename: /bin/ls
flavour: 5
endianess: 1
architecture: i386:x86-64
linkage: dynamic
interpreter: /lib64/ld-linux-x86-64.so.2
debug symbols: stripped

$ ./min_file ./test
filename: ./test
flavour: 5
endianess: 1
architecture: i386:x86-64
linkage: dynamic
interpreter: /lib64/ld-linux-x86-64.so.2
debug symbols: not stripped
```

BFD initialization

```
bfd *bfd_h = NULL;


bfd_init();


bfd_h = bfd_openr(filename, NULL);
if (!bfd_h)
    DIE("(openr) (%s)", bfd_errmsg(bfd_get_error()));


/* Is it a binary object?  */
if (!bfd_check_format(bfd_h, bfd_object))
    DIE("(check_format) (%s)", bfd_errmsg(bfd_get_error()));


/* Do we know how to read this?  */
if (bfd_get_flavour(bfd_h) == bfd_target_unknown_flavour)
    DIE("(get_flavour) (%s)", bfd_errmsg(bfd_get_error()));


render(bfd_h);
```

Print filename, flavour, architecture and endianess

```
    fprintf(stderr, "filename: %s\n", target->filename);
    fprintf(stderr, "flavour: %d\n", target->xvec->flavour);
    fprintf(stderr, "endianess: %d\n", target->xvec->byteorder);

    const bfd_arch_info_type *bfd_info = bfd_get_arch_info(target);
    fprintf(stderr, "architecture: %s\n", bfd_info->printable_name);
```

Flavour

```
enum bfd_flavour
{
  /* N.B. Update bfd_flavour_name if you change this.  */
  bfd_target_unknown_flavour,
  bfd_target_aout_flavour,
  bfd_target_coff_flavour,
  bfd_target_ecoff_flavour,
  bfd_target_xcoff_flavour,
  bfd_target_elf_flavour,
  bfd_target_ieee_flavour,
  bfd_target_nlm_flavour,
  ...
};
```

Dynamically linked vs static

- Dynamically linked executables have a section .interp
- This section contains the program interpreter (the dynamic loader)
- If this section is missing the the executable is not dynamically linked

Find the section .interp

```
/* Check if dynamic (section .interp should exist).  */
   if (find_section(target, ".interp")) {
       fprintf(stderr, "linkage: dynamic\n");
       render_interpreter(target);
   } else
       fprintf(stderr, "linkage: static\n");
```

find_section()

```
unsigned int find_section(bfd * target, const char * section_name) {

    asection * bfd_sec;

    for (bfd_sec = target->sections; bfd_sec; bfd_sec = bfd_sec->next) {
        if (!strcmp(bfd_sec->name, section_name))
            return 1;
    }

    return 0;
}
```

Interpreter name

```
void render_interpreter(bfd * target) {
    /* If it is a dynamically linked executable
       print the interepreter.  */
    asection * bfd_sec, * interp_section = NULL;

    for (bfd_sec = target->sections; bfd_sec; bfd_sec = bfd_sec->next) {
        if (!strcmp(bfd_sec->name, ".interp"))
            interp_section = bfd_sec;
    }

    int size = bfd_section_size(target, interp_section);
    char *interpreter = malloc(size);
    bfd_get_section_contents(target, interp_section, interpreter, 0, size);
    fprintf(stderr, "interpreter: %s\n", interpreter);


    free(interpreter);
}
```

Stripped vs non stripped

- Non stripped binaries have debugging symbols
  - They have a symbol table
  - The symbol table is stored in section .symtab
- libbfd does not print all sections
  - Obsolete API
  - Verify this with objdump –h
- We attempt to read the symbol table
  - If we can read it, then the binary is not stripped

Attempt to read the symbol table

```
long n = bfd_get_dynamic_symtab_upper_bound(target);
asymbol **bfd_symtab = malloc(n);
long nsyms = bfd_canonicalize_symtab(target, bfd_symtab);
if (nsyms == 0) {
    fprintf(stderr, "debug symbols: stripped\n");
} else
    fprintf(stderr, "debug symbols: not stripped\n");

free(bfd_symtab);
```

Homework

- Beautify the output of min_file.c
- Use it with different binaries
  - You may spot bugs!
- Add more features
  - Gray/shaded text in slide 6