Lab 7: ELF-introduction

This lab may be done either solo or in pairs.

In the following labs, you will learn to handle object and executable files. We will begin by learning just some of the basics of the ELF file format, and applications you can already use at this level - editing binary files and writing software patches.
Then, we will continue our study of the ELF format, by beginning to parse the structures in the ELF file, and use them for various purposes. In particular, we will access the data in the section header table and in the symbol table.

Lab goals

1. Extracting useful information from files in the ELF format.
2. Fixing files using this information: reverse engineering.

Methodology

- Get to know the ELF.
- Learn how to use the "readelf" utility. By using "readelf " you can get in a human readable format all the ELF structural information.
- Experience basic ELF manipulation.

Recommended operating procedure

This advice is relevant for all tasks. Note that while at some point you will no longer be using "hexedit" to process the file and "readelf" toget the information, nevertheless in some cases you may still want to use these tools for debugging purposes. In order to take advantage of these tools and make your tasks easier, you should:

- Print debugging messages: in particular the offsets of the various items, as you discover them from the headers.
- Use hexedit and readelf to compare the information you are looking for, especially if you run into unknown problems: hexedit is great if you know the exact location of the item you are looking for.
- Note that while the object files you will be processing will be linked using ld, and will, in most cases, use direct system calls in order to make the ELF file simpler, there is no reason why the programs you write need use this interface. You are allowed to use the standard library when building your own C programs.
- In order to preserve your sanity, even if the code you MANIPULATE may be without stdlib, we advise that for your OWN CODE you DO use the C standard library! (Yes this is repeated twice, so that you notice it!)
- In order to keep sane in the following labs as well, **understand** what you are doing and **keep track** of that and of your code, as you will be using them in future labs.

Lab 7 tasks

Note: All the executable files we will work with in this session are 32-Bit ELF binaries.
Note: Some numbers are in decimal and some are in hexadecimal. Pay careful attention to which is which.

Task 0

Task 0a

Download the following file: a.out. Answer the following questions (be prepared to explain your answers to the lab instructor):

1. Where is the entry point specified, and what is its value?
2. How many sections are there in this a.out?
3. What is the size of .text section?
4. Does the symbol _start occur in the file? If so, where is it mapped to in virtual memory?
5. Does the symbol main occur in the file? If so, where is it mapped to in virtual memory?
6. Where in the file does the code of function "main" start?

Task 0b

Write a program `hexeditplus` which receives a single command-line argument:

```
./hexeditplus filename
```

The file `filename` is the file to be used by the program for various operations, to be defined below.

First, define a menu for the user with a number of predefined functions (as done in Lab 2), to which we will add functions as we go.
For example, if functions: Display, Modify, and Quit are available, then the command line:

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Modify
3-Quit
```

For this part use an array with the above menu names and pointers to a function that calls exit(0) to quit the program.
The rest of the functions will be written in the next tasks.
If `filename` cannot be opened for reading and writing print an error message and exit.

Each function should get `filename` as a parameter.

Task 1: hexeditplus

In this task we will write our own version of hexedit for working with binary files. Note: You should verify that there is no error when opening a file. In case of an error, you should print a message and exit.

Task 1a: Display

Write the function for the "Display" option, which works as follows:

- Prompts the user for two numbers: `length` and `location`. (remember to use fgets and then `sscanf`, rather than `scanf` directly.)
- Opens `filename` for read.
- Displays in hexadecimal `length` bytes from the file starting from file position `location`. The "Modify" option works exactly the same as "Modify" option in task 0b.

For example, the command

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Modify
3-Quit
```

If the user chooses 1, `length` and `location` are read from the user.
For the numbers: 10 303 the program should open the file "abc" and print the 10 bytes, from byte 303 to byte 312 in the file.
The output should look like:

```
00 01 00 00 00 2F 6C 69 62 2F
```

You should use hexedit to verify that your code works correctly.
Here is some of hexedit's output for the file abc, verify that you understand why the output is as it is (remember that 303 in decimal is 0x12F in hexadecimal).

```
00000070   01 00 00 00  01 00 00 00  00 00 00 00  00 80 04 08  ................
00000080   00 80 04 08  EC 05 00 00  EC 05 00 00  05 00 00 00  ................
00000090   00 10 00 00  01 00 00 00  14 0F 00 00  14 9F 04 08  ................
000000A0   14 9F 04 08  0C 01 00 00  14 01 00 00  06 00 00 00  ................
000000B0   00 10 00 00  02 00 00 00  28 0F 00 00  28 9F 04 08  ........(...(...
000000C0   28 9F 04 08  C8 00 00 00  C8 00 00 00  06 00 00 00  (...............
000000D0   04 00 00 00  04 00 00 00  48 01 00 00  48 81 04 08  ........H...H...
000000E0   48 81 04 08  44 00 00 00  44 00 00 00  04 00 00 00  H...D...D.......
000000F0   04 00 00 00  51 E5 74 64  00 00 00 00  00 00 00 00  ....Q.td........
00000100   00 00 00 00  00 00 00 00  00 00 00 00  06 00 00 00  ................
00000110   04 00 00 00  52 E5 74 64  14 0F 00 00  14 9F 04 08  ....R.td........
00000120   14 9F 04 08  EC 00 00 00  EC 00 00 00  04 00 00 00  ................
00000130   01 00 00 00  2F 6C 69 62  2F 6C 64 2D  6C 69 6E 75  ..../lib/ld-linu
00000140   78 2E 73 6F  2E 32 00 00  04 00 00 00  10 00 00 00  x.so.2..........
00000150   01 00 00 00  47 4E 55 00  00 00 00 00  02 00 00 00  ....GNU.........
00000160   06 00 00 00  0F 00 00 00  04 00 00 00  14 00 00 00  ................
00000170   03 00 00 00  47 4E 55 00  C1 4E 4D 18  B9 A6 21 8F  ....GNU..NM...!.
```

Remember - the function should get the filename as a parameter, not an open file descriptor

Task 1b: Modify

Write the function for the "Modify" option:
This option opens the file `filename` for modification, and replaces the byte at `location` with the byte `val`.
The steps are:
a. Prompt the user for `location` and `val` (in hexadecimal).
b. Open `filename` appropriately and replace the byte at `location` in the file with `val`.
For example, the command line:

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Modify
3-Quit
```

After the user chooses 2 with `location` 303 and `val` "2D" - the value of that byte will be 0x2D.
Check this with the running example from task 1a, the output should change to :

```
2D 01 00 00 00 2F 6C 69 62 2F
```

Task 1c: Copy from file

Write the function for the "Copy from file" option:
This option replaces `length` bytes at `target-location` of `filename` with bytes from the `source-location`.

For example, the command

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Modify
3-Copy from file
4-Quit
```

When the user chooses option 3 , the program should query the user for:
`source-file` name,
`source location` (source file offset, in hexadecimal),
`target location` (target file offset, in hexadecimal)
`length` (number of bytes, in decimal).
For example, chosing option "3-Copy from file" using file `afile`, `source-location` 102, `length` 4 and `target-location` 33,
the program should read `length` = 4 bytes of file `afile` starting at offset `0x102`
and write them to the file `abc` starting from offset `0x33` (overwriting what was originally there).
Note that the target file is always specified by the file name given as the command-line argument to hexeditplus, both here and in task 1d.
Also observe that that after you execute this option, **only** `length` bytes of the file `targetfile` should be changed.
Use hexedit to demonstrate to the lab instructor that your code works.

Task 1d: Copy from memory

This option replaces `length` bytes at `target-location` of `filename` with bytes from `source-location`.
For this option the source is the main memory of the current program (i.e. hexeditplus).

For example, the command

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Modify
3-Copy from file
4-Copy from mem
5-Quit
```

When the user chooses option 4, the program should query the user for `source location` (a memory address in hexadecimal), a `target location` (file offset in hexadecimal), and `length` (number of bytes in decimal).
Then the program should open the target file `filename` for modification, and replace `length` bytes starting from `offset` with the bytes in memory starting at `source location`.

Task 1e

If you are registered to the architecture and SPlab course, do not do this task (you will have a different task later). This task is only for people who do not study assembly language at present. This task may be done in a completion lab.

Add to `hexeditplus` a "Display-ascii" function. This should be the same as "Display" except that the bytes are printed in the ascii format of hexedit (printable characters are printed as characters, and the others are printed as dots).
For example:

```
./hexeditplus abc
```

Will print:

```
File: abc, choose action:
1-Display
2-Display(ascii)
3-Modify
4-Copy from mem
5-Copy from file
6-Quit
```

Upon choice of Display(ascii) with location of 303 and length of 10 the expected output is:

```
...../lib/
```

Task 2: reading ELF

Task 2a

Download the following files: chezi, originalchezi.

chezi and **originalchezi** are both executable files in ELF format. They are almost the same except chezi behaves differently from originalchezi. Your task is to understand the reason for that.

Do the following:

1. Run the files.
2. Do they differ in size?
3. Why does their output differ? (hint: use readelf -h)
4. In what way is the entry point represented in the ELF file (arrangement of bytes) ?

Task 2b

Use your hexeditplus program from task 1a to display the entry point of a file (choose Display with the right location/length).

```
./hexeditplus filename
```

What are the values of location/length? How do you know that?

Use the edit function form hexeditplus program to fix the `chezi` file so that it will behave like `originalchezi`.

Task 3: delving deeper into the ELF structure

Task 3a

The goal of this task is to display the compiled code (in bytes) of the `main` function in the `abc` executable above.

In order to do that, you need to (a) find the offset (file location) of the main function. (b) find the size of the main function. (c) use your `hexeditplus` program to display the content of the main function to the screen.

Finding the needed information:

1. Find the entry for the main function in the symbol table of the ELF executable (readelf -s).
2. In that reference you will find both the size of function and the function's virtual address and section number.
3. In the section table of the executable, find the entry for the function's section (readelf -S).
4. Find both the section's virtual address (Addr) and the section's file offset (Off).
5. Use the above information to find the file offset of the function. (the function file offset equals the section's file offset plus the function's virtual address minus the section's virtual address. Be prepared to explain why.)

Task 3a1

This task is only for students of the architecture and splab course.

What are the first two assembly instructions in the `main` function?
You cab use the opcode information in the nasm manual.

Task 3b

The following file ntsc was meant to be a palindrome recognizer. Download it, and run it in the commandline.

```
./ntsc aabbababaacca
./ntsc 1112111
```

What is the problem with the file? (*hint*, try this string: 12345678912345987654321)

Add a correct palindrome identification function (should get a char* and return an int) to the hexeditplus program.

Use the `hexeditplus` program to replace the buggy "is_pal" function in the "ntsc" file with the corrected version (remeber to compile with the -m32 flag in order to produce a 32bit binary file).
Do it using the Copy command from the (from the binary file itself, option 5-Copy from file).
[think: are there any kinds of restrictions on the code you write for the is_pal function?]

Explain how you did it, and show that it works.

Deliverables

As usual, you should do task 0 before the lab, as part of your preparation for the lab. In order to get full credit, you should complete at least up to task 3 (not inclusive) during the lab. Task 1e can also be done in a competion lab.
