Assignment 1 - BASC

BINARY ANALYSIS AND SECURE CODING - UniGE

Kevin Cattaneo - S4944382

Index

Extract

The goal of this assignment is to get acquainted with the ELF file format. Unzip ELF_files.zip, and you'll have a bunch of files to work with. All of them hide one flag, that is, a string of the form BASC{...}, where "..." is at least eight characters long. In particular, BASC{3T0N5}, which is something you may come across, is not the intended flag for the corresponding file.
Given that, I will `unzip ELF_files.zip`.

ELF 1

First of all I run a bunch of commands to make sure of what I'm handling.

```
file elf1
elf1: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=cb565ed38996d33e5c6966bbe1699175b9924c75, for
GNU/Linux 3.2.0, stripped
```

>> It seems like a plain x86-64 ELF executable.

```
./elf1
Nothing to see here…
```

`strings elf1` – no flags found

Then I explore the binary with `hte elf1`
and discover the BASC{3TON5}, that is not intended to be a valid flag.

Solution

Run the `readelf` command, just after a very big look to the man. The flag can be found with different command-flag combinations.
`readelf -a elf1` – shows everything that will follow
`readelf -S elf1`

```
There are 43 section headers, starting at offset 0x3158:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [17] B                 PROGBITS         00000000000011e5
000011e5
       000000000000000b  0000000000000000  AX       0     0     1
  [18] A                 PROGBITS         00000000000011f0
000011f0
       000000000000000b  0000000000000000  AX       0     0     1
  [19] S                 PROGBITS         00000000000011fb
000011fb
       000000000000000b  0000000000000000  AX       0     0     1
  [20] C                 PROGBITS         0000000000001206
00001206
       000000000000000b  0000000000000000  AX       0     0     1
  [21] {                 PROGBITS         0000000000001211
00001211
       000000000000000b  0000000000000000  AX       0     0     1
```

```
  [22] s                 PROGBITS         000000000000121c
0000121c
       000000000000000b  0000000000000000  AX       0     0     1
  [23] 3                 PROGBITS         0000000000001227
00001227
       000000000000000b  0000000000000000  AX       0     0     1
  [24] c                 PROGBITS         0000000000001232
00001232
       000000000000000b  0000000000000000  AX       0     0     1
  [25] T                 PROGBITS         000000000000123d
0000123d
       000000000000000b  0000000000000000  AX       0     0     1
  [26] i                 PROGBITS         0000000000001248
00001248
       000000000000000b  0000000000000000  AX       0     0     1
  [27] 0                 PROGBITS         0000000000001253
00001253
       000000000000000b  0000000000000000  AX       0     0     1
  [28] N                 PROGBITS         000000000000125e
0000125e
       000000000000000b  0000000000000000  AX       0     0     1
  [29] 5                 PROGBITS         0000000000001269
00001269
       000000000000000b  0000000000000000  AX       0     0     1
  [30] }                 PROGBITS         0000000000001274
00001274
       000000000000000b  0000000000000000  AX       0     0     1
  [31] .fini             PROGBITS         0000000000001280
00001280
       000000000000000d  0000000000000000  AX       0     0     4v


readelf -l elf1:
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 .rela.dyn .rela.plt
   03     .init .plt .plt.got .plt.sec .text B A S C { s 3 c T i 0
N 5 } .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .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      .init_array .fini_array .dynamic .got
```

Multiple sections at segment 03 have been created with their headers (so that we can see them from both Section and Program Headers), that compose the 8-length content of the flag: BASC{s3cTi0N5}, thanks for the README hint.

After a little investigation, the fact that BASC{3T0N5} is a subset of the real flag is because of the section ".shstrtab", that is a section that maps other sections and re-utilize bytes when certain elements (e.g. letters) are double used; indeed if we look at the offsets pointed by the content of this section we can also find the missing letters.

ELF 2

```
file elf2
elf2: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),
statically linked, for GNU/Linux 2.6.26,
BuildID[sha1]=84f38995563fddf7e782f901ad6c84a63c7e01f9, stripped
```
>> It seems to be like an ARM executable

```
./elf2
zsh: exec format error: ./elf2
```

`strings elf2` – no flag found

In the first place I tried to edit with hte editor to change some values in the ELF header to resemble a x86_64 and also tried to change endianness, but with no result.
With further exploration I verified that it is indeed an ARM, since we have some legit ARM sections in it.

```
readelf -l elf2
Section to Segment mapping:
  Segment Sections...
   00     .ARM.exidx
   01     .note.ABI-tag .note.gnu.build-id .init .text
__libc_freeres_fn .fini .rodata __libc_subfreeres __libc_atexit
.ARM.extab .ARM.exidx .eh_frame
   02     .tdata .init_array .fini_array .jcr .data.rel.ro .got
.data .bss __libc_freeres_ptrs
   03     .note.ABI-tag .note.gnu.build-id
   04     .tdata .tbss
   05
```

Solution

After reviewing the given material, I installed the qemu-user-binfmt virtualizer to run ARM.
Just launch elf2 again:

```
./elf2
BASC{ARMed_&_d4ng3r0uS}
```

ELF 3

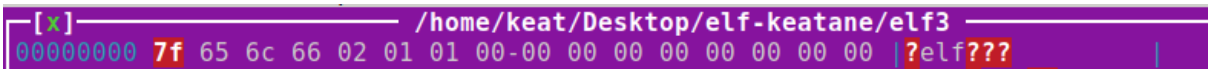
```
file elf3
elf3: data
```
>> This differs a lot from the other ELF files, in some sense it cannot recognize it is an ELF at all.

```
./elf3
zsh: exec format error: ./elf3
```
>> As expected it will not run

`strings elf3` – no flag found
But I discovered from this string "/lib64/ld-linux-x86-64.so.2", that the file I'm working with may be some kind of ELF 64-bit executable for x86_64 and not a generic "data" file.

Solution

After viewing the file with hte, I discover that the first identifier bytes for the magic number of the ELF type are different: the letters “elf” are in lowercase.



By changing them to the correct ones: 7f 45 4c 46 I get a correct ELF executable.

```
./elf3
BASC{cAs3_maTT3rS}
```

Another way is by running:
```
readelf -l elf3
readelf: Error: Not an ELF file - it has the wrong magic bytes at the start
```

ELF 4

```
file elf4
elf4: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked,
BuildID[sha1]=b7c34e2c255427db1a4bf366a804fc3d67ff35f0, for
GNU/Linux 3.2.0, stripped
```

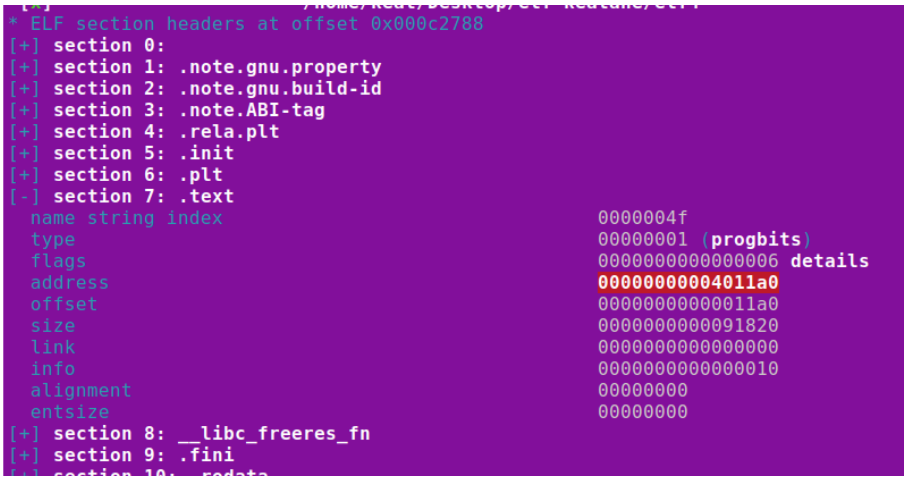
>> It seems a plain x86-64 ELF 64-bit executable

```
./elf4
[1]    6588 segmentation fault (core dumped)  ./elf4
```

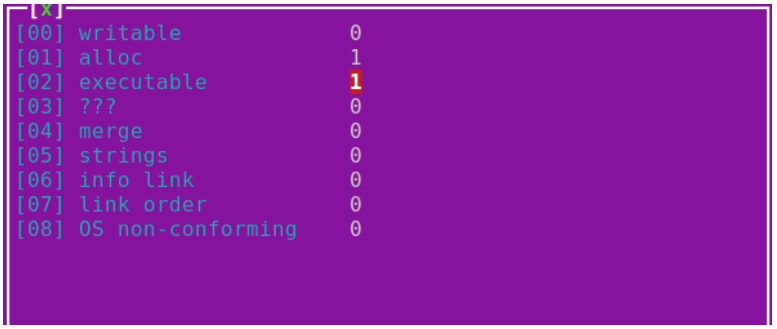
`strings elf4` – there is the universe made into strings in this program, but no flag found

I tried to edit with hte editor some ELF header bytes with different combinations, but the ELF always tells about a segmentation fault or about an exec format error after those edits.

After a lot of trial and error, I investigated with gdb debugger and I see the program stops immediately with at the very start (with both start and starti) at address:
`0x401c60  endbr64.`
Because of:
`Program terminated with signal SIGSEGV, Segmentation fault.`
This address is in the `.text` section, which holds the code of the program.



I check for details of `.text`, to make clear that everything is as it should be, in particular about correctness of permissions, specifically if it is executable. It is indeed.



Solution

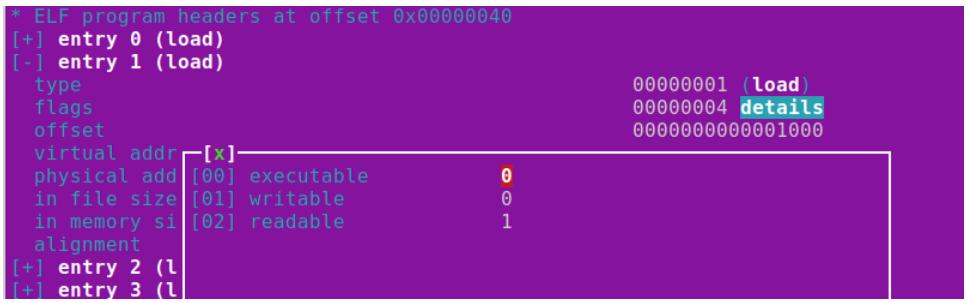
After checking the `.text` section I also tried to look at the program header table to see also details about the segment that contains the `.text` section.

```
readelf -l elf4
Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.property .note.gnu.build-id .note.ABI-tag
.rela.plt
   01     .init .plt .text __libc_freeres_fn .fini
   02     .rodata .stapsdt.base .eh_frame .gcc_except_table
   03     .tdata .init_array .fini_array .data.rel.ro .got
.got.plt .data __libc_subfreeres __libc_IO_vtables __libc_atexit
.bss __libc_freeres_ptrs
   04     .note.gnu.property
   05     .note.gnu.build-id .note.ABI-tag
   06     .tdata .tbss
   07     .note.gnu.property
   08
   09     .tdata .init_array .fini_array .data.rel.ro .got
```

I see that the `.text` section is in the segment 01, that is this segment:

```
Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags
Align
LOAD           0x0000000000001000 0x0000000000401000
0x0000000000401000
                 0x000000000009366d 0x000000000009366d  R
0x1000
```

To verify the correctness I compared that segment with the one correspondent (so the one that contains `.text`) in a plain x86_64 ELF executable (a simple printf in a main) of mine: I observed that the elf4 is missing the E(xecutable) permission on the segment above. This is also visible from hte.



So by editing this permission flag executable to 1 I repaired elf4.

```
./elf4
BASC{no_eXec_no_party}
```

Extra

“Flags are not four, but every file has one flag”.

If you check:

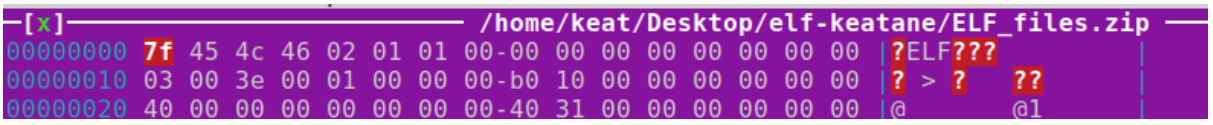
```
file ELF_files.zip
ELF_files.zip: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=69e1fd0c892100efe04b1cdb0628433d7bd26038, for
GNU/Linux 3.2.0, stripped
```

>> But wait, it should have been just a ZIP! So it is a polyglot file.

So if I run it:

```
chmod +x ELF_files.zip
./ELF_files.zip
BASC{can_U_run_a_ZIP?}
```

Indeed, if we explore the first part of the ZIP files, it has the ELF identifier.



The doubt that can arise is how: as we have seen file extensions are immaterial and, as course notes says:
“Different parsers can interpret the same sequence differently

- ZIP parsers look for the “End of Central Directory” from the **end** of the file
- ELF parsers expects an header at the **beginning**”