Elf from scratch

I’m learning about ELF, the binary file format for linux, BSD and others. As we will see, it is full of interesting information and will server as a good introduction to assembly programming.

Why I am interested in ELF?

I am currently looking for a full-time job with astring focus on cybersecurity. Form what I’ve read on job offers, low level debugging is a very after skill. That’s what I’m setting out to learn.

One low level primititive we use…

Understanding the ELF

I wrote an ELF binary which, when run, prints itself on standard output

It does’s cheat by reading its own file. Instead, it uses how elf binary files are loaded into memory. To explain how the program works. I’ll show you it byte-by-byte. Along the way, we’ll learn about files, programs, the ELF format, and tools for working with these things.

Files and hexadecimal

The program is just a file. This means it is a named list of bytes, just like text files, images, video, etc. we can view

ELF Header

The ELF header is always located at the beginning of the ELF file. It describes the ELF file organization and contains the actual size of the object file control structures.

The ELF header consists of the following fields, or membets, some have the value 0 for relocatables files:

e\_ident

this is a byte array consisting of the EI\_NIDENT initial bytes of the ELF header, where EI\_NIDENT is a name for 16. The elements of this array mark the file as an ELF object file and provide machine-independent data that can be used to decode and interpret the file’s contents.

E\_type

Identifies the object file type. A value of 1, that has the name EL\_REL, specifies a relocatable file. Describes all the object file types.

E\_machine

Specifies the required architecture for an individual file. A value of 3, that has the name EM\_386, specifies intel 80386. EM\_486, specifies intel 80486.

E\_version

Identifies the version of this object files’s format. This field should have the current version number, named EV\_CURRENT

E\_entry

Virtual address where the process is to start. A value of 0 indicates no associated entry point

E\_phoff

Program header table’s file offset, in bytes. The value of 0 indicates no program header. (Relocatable files do not need a program header tables.)

E\_shoff

Section header tables’s file offset, in bytes. The value of 0 indicates no section header table. (relocatable files must have a section header table.)

E\_flag

Processor-specific flags associated with the file. For the intel 80386, this field has value 0.

E\_ehsize

ELF header’s size, in bytes

E\_phentsize

Size, in bytes, of entries in the program header table. All entries are the same size. (relocatable files do not need a program header table.)

E\_phnum

Number of entries in program header table. A value of 0 indicates the file has no program header table.

E\_shentrize

Number of entries in program header table. A value of 0 indicates the file has no program header table.

E\_shnum

Number of entries in section header table. A value of 0 indicates the file has not section

Header table.

E\_shstrndx

Section header table index of the entry associated with the section name string table. A value of SHN\_UNDEF indicates the file does not have a section name string table