CSC 6580
Spring 2020

Instructor: Stacy Prowell

Executable File Formats: Executable and Linking Format (ELF)

ELF

The ELF format is defined in
`/usr/include/elf.h`

Defines structs, flags, types, etc.

The ELF format is documented in:
```
$ man elf
```

Explains the structs, flags, types, etc.

ELF

You can see the content of an ELF file using the `readelf` command.

You can modify the content of an ELF file using the `elfedit` command.

But you probably shouldn’t. Well… unless...

ELF

More? Visit the Hello World ELF Tutorial:
http://www.cirosantilli.com/elf-hello-world/

pyelftools
https://github.com/eliben/pyelftools/wiki/User's-guide

A Bit About Assembly

Resources

- ODA https://onlinedisassembler.com/odaweb
- X86 Opcode and Instruction Reference https://ref.x86asm.net
- Intel Documentation (search Intel 64 and IA-32 Architectures Software Developer’s Manual)
- The incomplete reference https://felixcloutier.com/x86

Assembly

Compilers convert programs into an *intermediate representation (IR)*. This is just a form that is useful or convenient for the compiler and for subsequent tools. For instance, GCC converts C and other languages into a *register transfer logic (RTL)*. The processor doesn't natively understand C… or RTL.

This IR is converted into *assembly language*. There is still a lot to be decided with these files; we'll see more of that later in the course. The processor doesn't natively understand assembly language, either.

Assembly is not one-to-one with *machine code*, which is what the processor natively understands.

A Few Types of Instructions

- Programs are stored in memory, and a *program counter* keeps track of the next instruction to fetch, decode, and execute.
- An instruction can:
  - Read the processor state, read memory, write to memory, and modify the processor state
  - The instruction also modifies the program counter - typically it just points to the next instruction
  - On the Intel architecture, instructions can be up to 15 bytes in length

Branching

- “Straight line” instructions transform the processor state and continue to the next instruction in the program: `adc eax, 22`
- These instructions typically set *condition flags* based on the result of the computation (zero flag, sign flag, etc.)
- “Branching” instructions typically do not transform the processor state, but instead examine it and then:
    - Conditionally branch, typically based on the value of flags
      `jz +23`
      If the zero flag is set, then branch forward 23 bytes. If it is not set, then continue with the next instruction
    - Unconditionally branch
      `jmp 0x21e343ea`

Aside: Important Tools

There are a *lot* of these, but two you should be familiar with are:

- objdump
- hexdump

These are going to be on nearly any platform and you should get to know them.

Better disassemblers and full hex editors are also great...

Next Time:
Writing Assembly