Binary Exploitation

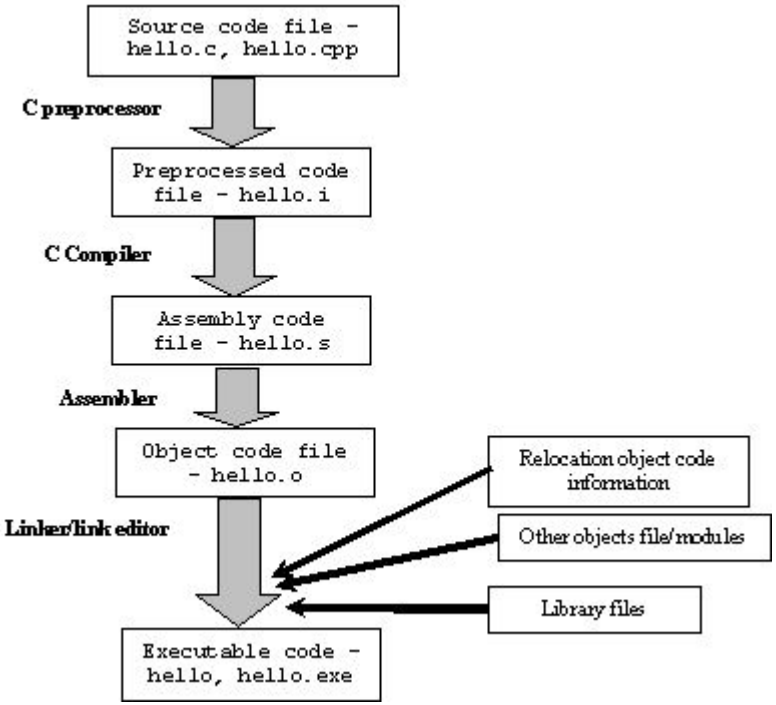
From C to Memory — 0x0

Roadmap

- Compilation Process
  - What
- Memory Layout
  - Where
- Runtime
  - How

Compilation Process

1. Preprocessing
2. Compilation
3. Assembly
4. Linking



1. Preprocessing

- Macro definitions
- File inclusion
- Conditional compilation
- Source file goes in, preprocessed file comes out
- Input and output are text

- #define NUMBER 1337
- #include <stdio.h>
- #ifdef (...) #endif
- gcc -E <input>.c

2. Compilation

- Parses the source code and produces assembly code
- Previous output goes in, compiled file comes out
- Input and output are text
- Output is Assembly code and architecture dependent

- gcc -S <input>.c
  - -masm=intel
  - -m32

3. Assembly

- Assembles the code into machine code
- Actual instructions to be run
- Input is text
- Output is binary

- gcc -c <input>.c

4. Linking

- Creates a runnable file from the previous object
- Links missing information from previous stage
  - Libraries
  - External functions
- Static or dynamic linking

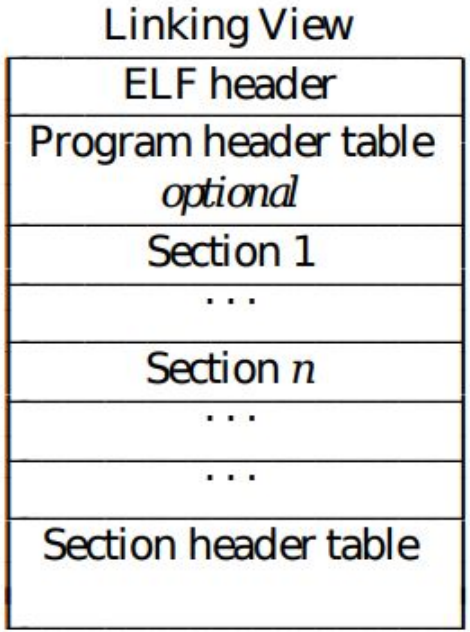
- gcc <input>.c
  - -m32

4. Linking

- Dynamic (default)
  - Creates references to external objects
  - Smaller file size
  - gcc <input>.c
    - -m32
- Static
  - No external references
  - Bigger file size
  - gcc --static <input>.c
    - -m32

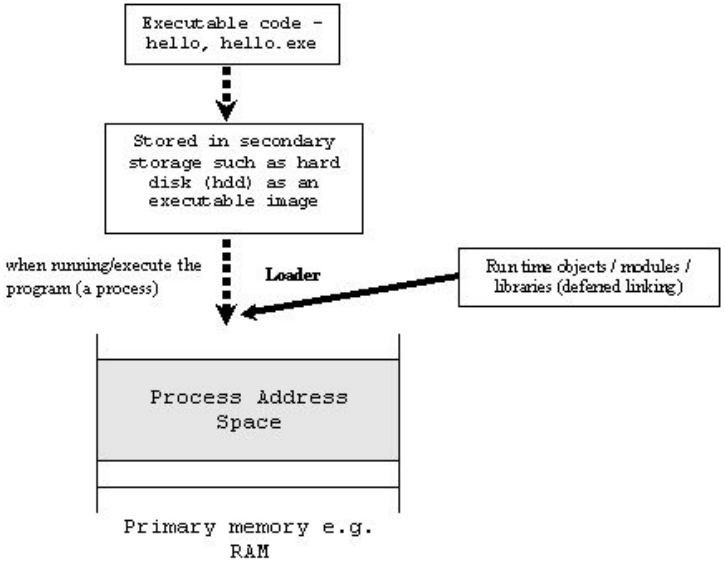
Executable and Linkable Format (ELF)

- Object is composed of sections
- Section Header Table (SHT) describes the sections
- readelf -S <file>



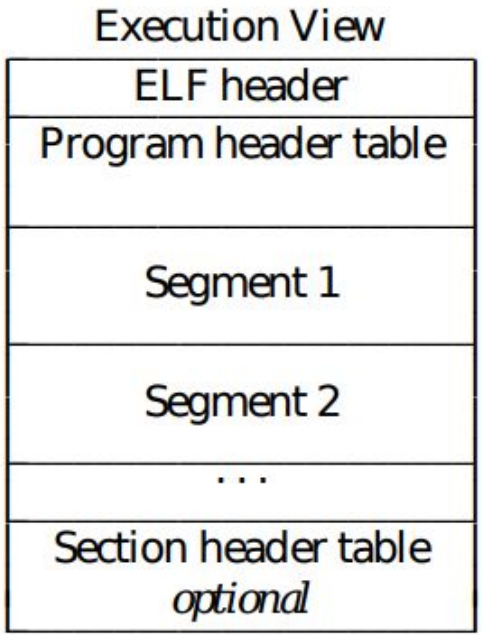
Memory Layout

1. Process is spawn
2. Memory is allocated
3. Permissions are set



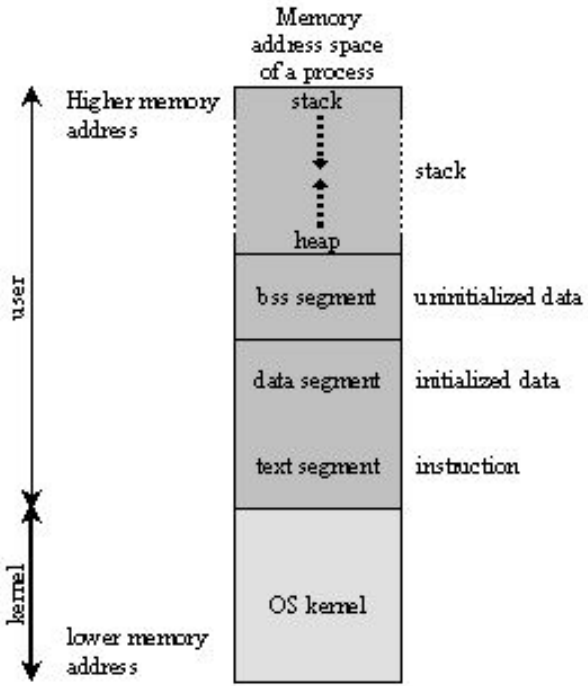
Memory Layout

- Sections are interpreted as segments
- Segments are flagged according to their purpose
  - read
  - write
  - Execute
- readelf -l <file>



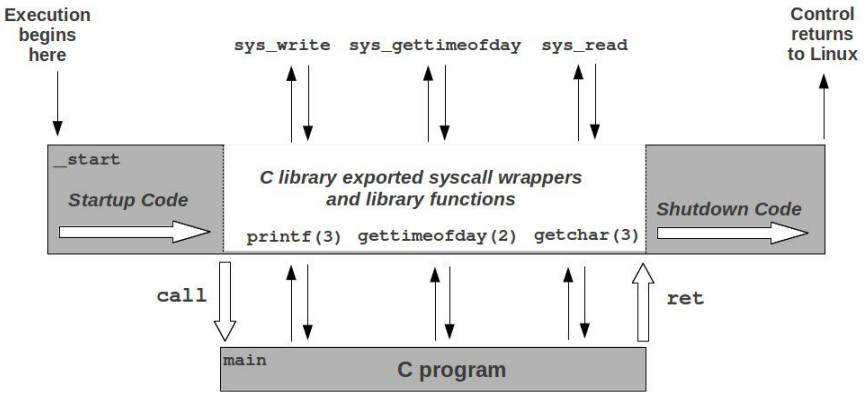
Memory Layout

- Segments, lower to upper
  - .text
  - .data
  - .bss
  - heap
  - stack
- cat /proc/<pid>/maps
- pmap <pid>



Runtime

1. _start is called
    a. Contains startup code
2. main is called
    a. Runs user code



Runtime

```
gcc -m32 -O0 -o lecture00 lecture00.c -Wl,-z,norelro
```

```c
#include <stdio.h>

int main() {

  puts("Hello");

  puts("I heard there's a shell in /bin/sh");

}
```

```
objdump -M intel -d lecture00
     <main>:
     <main prologue>
push   0x80484d0
call   80482e0 <puts@plt>
add    esp,0x10
sub    esp,0xc
push   0x80484d8
call   80482e0 <puts@plt>
     <main epilogue>
```

I’m gonna read this, eventually

http://www.tenouk.com/ModuleW.html

https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/

http://www.skyfree.org/linux/references/ELF_Format.pdf

http://nairobi-embedded.org/070_elf_c_runtime.html

https://systemoverlord.com/2017/03/19/got-and-plt-for-pwning.html

Next week...

- Interacting with binaries
  - CLI & Python
  - Locally & remotely
- Kali.ova provided