Australian
National
University

The ELF For

COMP3703 Software Security

Based on Chapter 2 of Andriesse's "Practical Binary Analysis"
Slides by H. Gunadi

Outline

- Overview of ELF
- Executable headers
- Sections and section headers
- Program headers
- Lazy binding

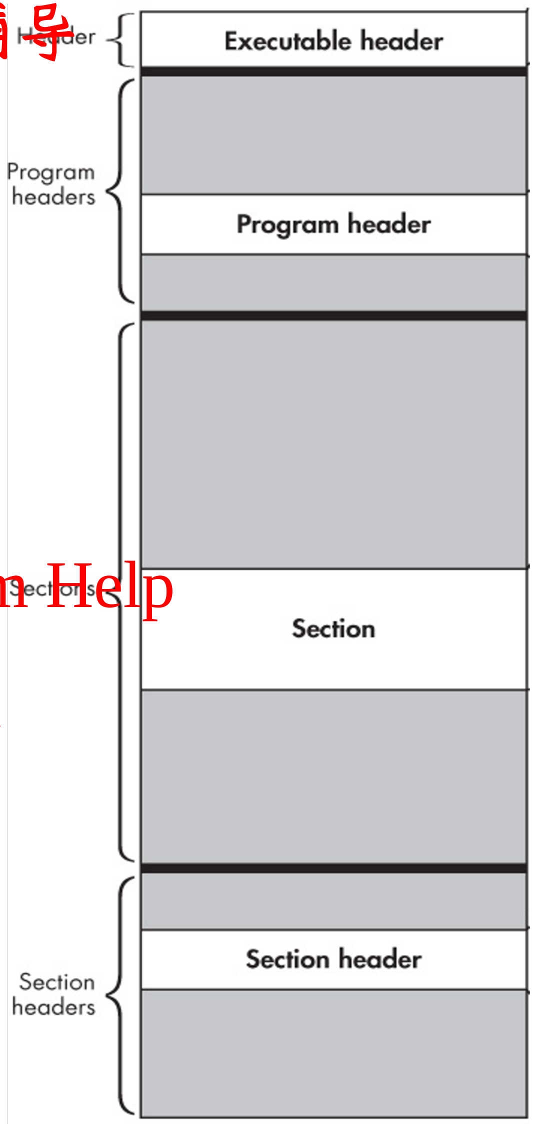
What is ELF?

- Executable and Format (ELF) is the default binary format on ased systems.

- Used for executable files, object files, shared libraries and core dumps.

Components of ELF

- We focus on Li [illegible] bit ELF, but 32-bit format is [illegible]
- Four types of components:
  - Executable headers
  - Program headers – needed for executable
  - Sections
  - Section headers (optional) – used by linker



Executable Header

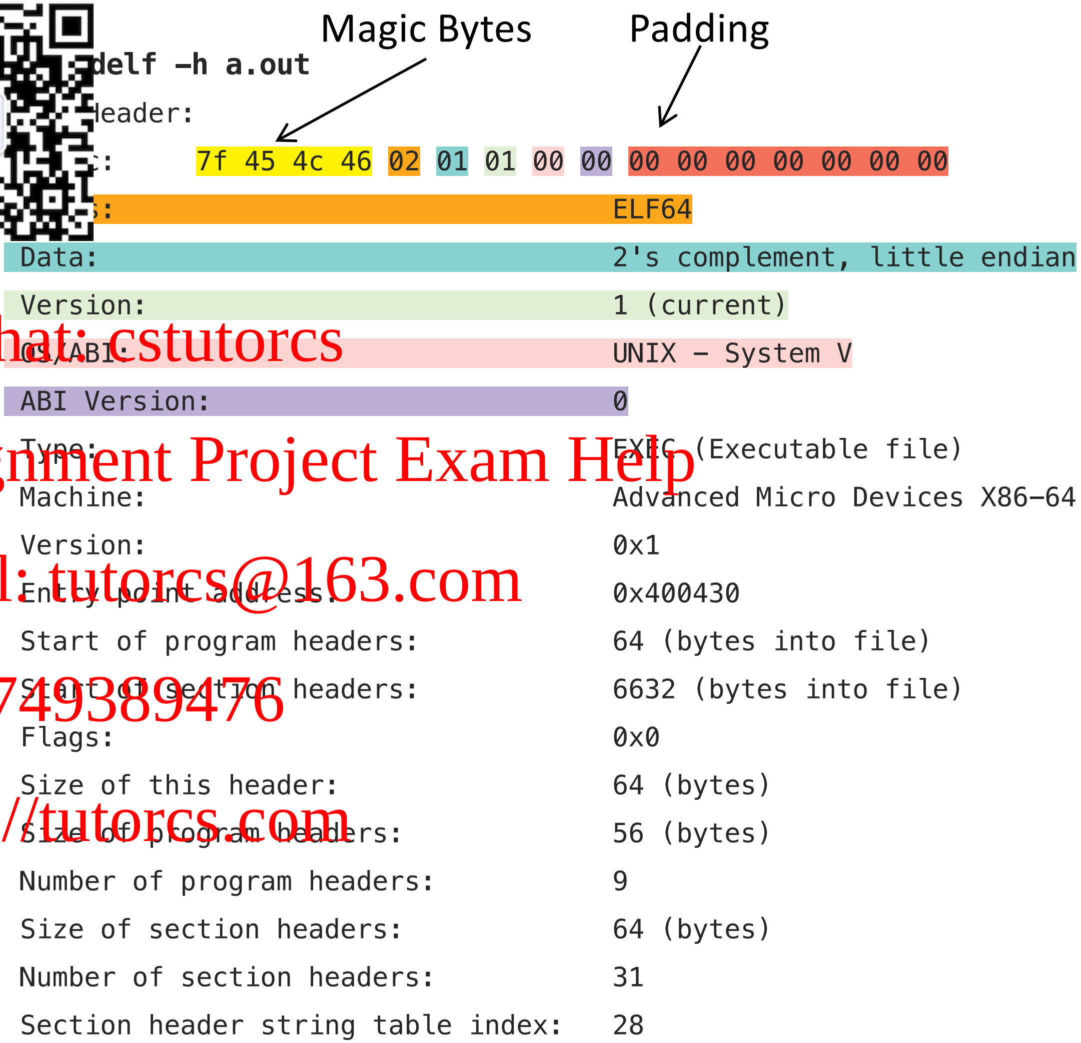
- Series of bytes to gi information about th binary, e.g., what kind of ELF file, where to find other contents of the file.

- Various definitions and constants in */usr/include/elf.h*

```
    ef struct {
    igned char e_ident[16]; /* Magic number and other info */
    t16_t   e_type;       /* Object file type */
    t16_t   e_machine;   /* Architecture */
 uint32_t   e_version;   /* Object file version */
 uint64_t   e_entry;     /* Entry point virtual address */
 uint64_t   e_phoff;     /* Program header table file offset */
 uint64_t   e_shoff;     /* Section header table file offset */
 uint32_t   e_flags;      /* Processor-specific flags */
 uint16_t   e_ehsize;    /* ELF header size in bytes */
 uint16_t   e_phentsize; /* Program header table entry size */
 uint16_t   e_phnum;     /* Program header table entry count */
 uint16_t   e_shentsize;/* Section header table entry size */
 uint16_t   e_shnum;     /* Section header table entry count */
 uint16_t   e_shstrndx; /* Section header string table index */
} Elf64_Ehdr;
```

Executable Header

```
typedef struct {
      unsigned char e_ident[16];
      uint16_t  e_type;
      uint16_t  e_machine;
      uint32_t  e_version;
      uint64_t  e_entry;
      uint64_t  e_phoff;
      uint64_t  e_shoff;
      uint32_t  e_flags;
      uint16_t  e_ehsize;
      uint16_t  e_phentsize;
      uint16_t  e_phnum;
      uint16_t  e_shentsize;
      uint16_t  e_shnum;
      uint16_t  e_shstrndx;
} Elf64_Ehdr;
```



```
$ readelf -h a.out
ELF Header:
  Magic:    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400430
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6632 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 28
```

Executable Header

```
typedef struct {
unsigned char e_ident[16];
uint16_t     e_type;
uint16_t     e_machine;
uint32_t     e_version;
uint64_t     e_entry;
uint64_t     e_phoff;
uint64_t     e_shoff;
uint32_t     e_flags;
uint16_t     e_ehsize;
uint16_t     e_phentsize;
uint16_t     e_phnum;
uint16_t     e_shentsize;
uint16_t     e_shnum;
uint16_t     e_shstrndx;
} Elf64_Ehdr;
```

ET_RE
ET_EX
ET_DY
etc.

EM_X86_64,
EM_386,
EM_ARM,
etc.

```
eadelf -h a.out
F Header:
gic:     7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
ass:                               ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x400430
Start of program headers:          64 (bytes into file)
Start of section headers:          6632 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         9
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 28
```

Executable Header

```
typedef struct {
unsigned char e_ident[16];
uint16_t    e_type;
uint16_t    e_machine;
uint32_t    e_version;
uint64_t    e_entry;
uint64_t    e_phoff;
uint64_t    e_shoff;
uint32_t    e_flags;
uint16_t    e_ehsize;
uint16_t    e_phentsize;
uint16_t    e_phnum;
uint16_t    e_shentsize;
uint16_t    e_shnum;
uint16_t    e_shstrndx;
} Elf64_Ehdr;
```

Usually 0 for x86_64

Section index into .shstrtab section, the names of all sections in the binary

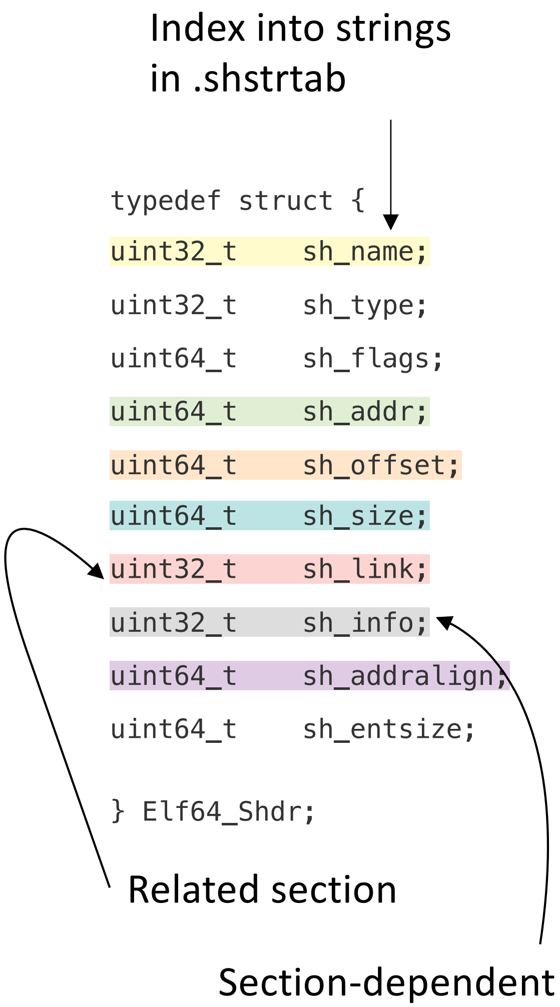
```
$ readelf -h a.out
 f Header:
 gic:     7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
 ss:                               ELF64
 ta:                               2's complement, little endian
 rsion:                            1 (current)
 OS/ABI:                           UNIX - System V
 ABI Version:                      0
 Type:                             EXEC (Executable file)
 Machine:                          Advanced Micro Devices X86-64
 Version:                          0x1
 Entry point address:              0x400430
 Start of program headers:         64 (bytes into file)
 Start of section headers:         6632 (bytes into file)
 Flags:                            0x0
 Size of this header:              64 (bytes)
 Size of program headers:          56 (bytes)
 Number of program headers:        9
 Size of section headers:          64 (bytes)
 Number of section headers:        31
 Section header string table index: 28
```

Sections and Section Headers

- Section structures vary depending on the contents.
- Described in Section Headers.
- Not all sections are used during execution.
- Section headers are optional, only for linking.

```
typedef struct {
    uint32_t    sh_name;    /* Section name (string tbl index) */
    uint32_t    sh_type;    /* Section type */
    uint64_t    sh_flags;   /* Section flags */
    uint64_t    sh_addr;    /* Section virtual addr at execution */
    uint64_t    sh_offset;  /* Section file offset */
    uint64_t    sh_size;    /* Section size in bytes */
    uint32_t    sh_link;    /* Link to another section */
    uint32_t    sh_info;    /* Additional section information */
    uint64_t    sh_addralign;/* Section alignment */
    uint64_t    sh_entsize; /* Entry size if section holds table */
} Elf64_Shdr;
```

Section Headers



Index into strings in .shstrtab

```
typedef struct {
uint32_t     sh_name;
uint32_t     sh_type;
uint64_t     sh_flags;
uint64_t     sh_addr;
uint64_t     sh_offset;
uint64_t     sh_size;
uint32_t     sh_link;
uint32_t     sh_info;
uint64_t     sh_addralign;
uint64_t     sh_entsize;

} Elf64_Shdr;
```

Related section

Section-dependent

```
$ readelf --sections --wide a.out

Th                  headers, starting at offset 0x19e8:

S

                         Type          Address          Off    Size   ES  Flg Lk Inf Al
                         NULL          0000000000000000 000000 000000 00      0   0  0
                         PROGBITS      0000000000400238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE          0000000000400254 000254 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE         0000000000400274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH      0000000000400298 000298 00001c 00   A  5   0  8
  [ 5] .dynsym           DYNSYM        00000000004002b8 0002b8 000060 18   A  6   1  8
  [ 6] .dynstr           STRTAB        0000000000400318 000318 00003d 00   A  0   0  1
  [ 7] .gnu.version      VERSYM        0000000000400356 000356 000008 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED       0000000000400360 000360 000020 00   A  6   1  8
  [ 9] .rela.dyn         RELA          0000000000400380 000380 000018 18   A  5   0  8
  [10] .rela.plt         RELA          0000000000400398 000398 000030 18  AI  5  24  8
  [11] .init             PROGBITS      00000000004003c8 0003c8 00001a 00  AX  0   0  4
  [12] .plt              PROGBITS      00000000004003f0 0003f0 000030 10  AX  0   0 16
  [13] .plt.got          PROGBITS      0000000000400420 000420 000008 00  AX  0   0  8
  [14] .text             PROGBITS      0000000000400430 000430 000192 00  AX  0   0 16
  [15] .fini             PROGBITS      00000000004005c4 0005c4 000009 00  AX  0   0  4
...
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Section Headers

```
typedef struct {
uint32_t     sh_name;
uint32_t     sh_type;
uint64_t     sh_flags;
uint64_t     sh_addr;
uint64_t     sh_offset;
uint64_t     sh_size;
uint32_t     sh_link;
uint32_t     sh_info;
uint64_t     sh_addralign;
uint64_t     sh_entsize;
} Elf64_Shdr;
```

Some important types:
SHT_STRTAB
SHT_SYMTAB
SHT_REL / SHT_RELA
SHT_DYNSYM
SHT_DYNAMIC
SHT_PROGBITS

```
$ readelf --sections --wide a.out
                    tion headers, starting at offset 0x19e8:



                           Type              Address          Off    Size   ES  Flg Lk Inf Al
                           NULL              0000000000000000 000000 000000 00      0   0  0
                           PROGBITS          0000000000400238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag       NOTE              0000000000400254 000254 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE               0000000000400274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash           GNU_HASH          0000000000400298 000298 00001c 00   A  5   0  8
  [ 5] .dynsym             DYNSYM            00000000004002b8 0002b8 000060 18   A  6   1  8
  [ 6] .dynstr             STRTAB            0000000000400318 000318 00003d 00   A  0   0  1
  [ 7] .gnu.version        VERSYM            0000000000400356 000356 000008 02   A  5   0  2
  [ 8] .gnu.version_r      VERNEED           0000000000400360 000360 000020 00   A  6   1  8
  [ 9] .rela.dyn           RELA              0000000000400380 000380 000018 18   A  5   0  8
  [10] .rela.plt           RELA              0000000000400398 000398 000030 18  AI  5  24  8
  [11] .init               PROGBITS          00000000004003c8 0003c8 00001a 00  AX  0   0  4
  [12] .plt                PROGBITS          00000000004003f0 0003f0 000030 10  AX  0   0 16
  [13] .plt.got            PROGBITS          0000000000400420 000420 000008 00  AX  0   0  8
  [14] .text               PROGBITS          0000000000400430 000430 000192 00  AX  0   0 16
  [15] .fini               PROGBITS          00000000004005c4 0005c4 000009 00  AX  0   0  4
...
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Section Headers

```
typedef struct {
uint32_t     sh_name;
uint32_t     sh_type;
uint64_t     sh_flags;
uint64_t     sh_addr;
uint64_t     sh_offset;
uint64_t     sh_size;
uint32_t     sh_link;
uint32_t     sh_info;
uint64_t     sh_addralign;
uint64_t     sh_entsize;
} Elf64_Shdr;
```

Size of structured contents

Some of the important flags:
SHF_WRITE, SHF_ALLOC, SHF_EXECINSTR

```
$ readelf --sections --wide a.out
Ther                      eaders, starting at offset 0x19e8:

Sect
  [N                      Type            Address          Off    Size   ES Flg Lk Inf Al
  [                       NULL            0000000000000000 000000 000000 00      0   0  0
  [                       PROGBITS        0000000000400238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag      NOTE            0000000000400254 000254 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE            0000000000400274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash          GNU_HASH        0000000000400298 000298 00001c 00   A  5   0  8
  [ 5] .dynsym            DYNSYM          00000000004002b8 0002b8 000060 18   A  6   1  8
  [ 6] .dynstr            STRTAB          0000000000400318 000318 00003d 00   A  0   0  1
  [ 7] .gnu.version       VERSYM          0000000000400356 000356 000008 02   A  5   0  2
  [ 8] .gnu.version_r     VERNEED         0000000000400360 000360 000020 00   A  6   1  8
  [ 9] .rela.dyn          RELA            0000000000400380 000380 000018 18   A  5   0  8
  [10] .rela.plt          RELA            0000000000400398 000398 000030 18  AI  5  24  8
  [11] .init              PROGBITS        00000000004003c8 0003c8 00001a 00  AX  0   0  4
  [12] .plt               PROGBITS        00000000004003f0 0003f0 000030 10  AX  0   0 16
  [13] .plt.got           PROGBITS        0000000000400420 000420 000008 00  AX  0   0  8
  [14] .text              PROGBITS        0000000000400430 000430 000192 00  AX  0   0 16
  [15] .fini              PROGBITS        00000000004005c4 0005c4 000009 00  AX  0   0  4
...
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Section: NULL, .init, and .fini

- NULL: a section w[illegible]try, name, nor bytes; used to mark the first section he[illegible]

- .init: Run before any other code in the binary is executed, akin to constructor in OOP. Executed before the main entry point in the program.

- .fini: Run after program completes, akin to destructor in OOP.

Section: .text

- Main code of the p[illegible]
- SHT_PROGBITS: [illegible] user-defined code.
- Executable but not writable.
- Usually the executable does not directly point to the main function, but through _start and __libc_start_main.
- __libc_start main resides in .plt section / part of a shared library.

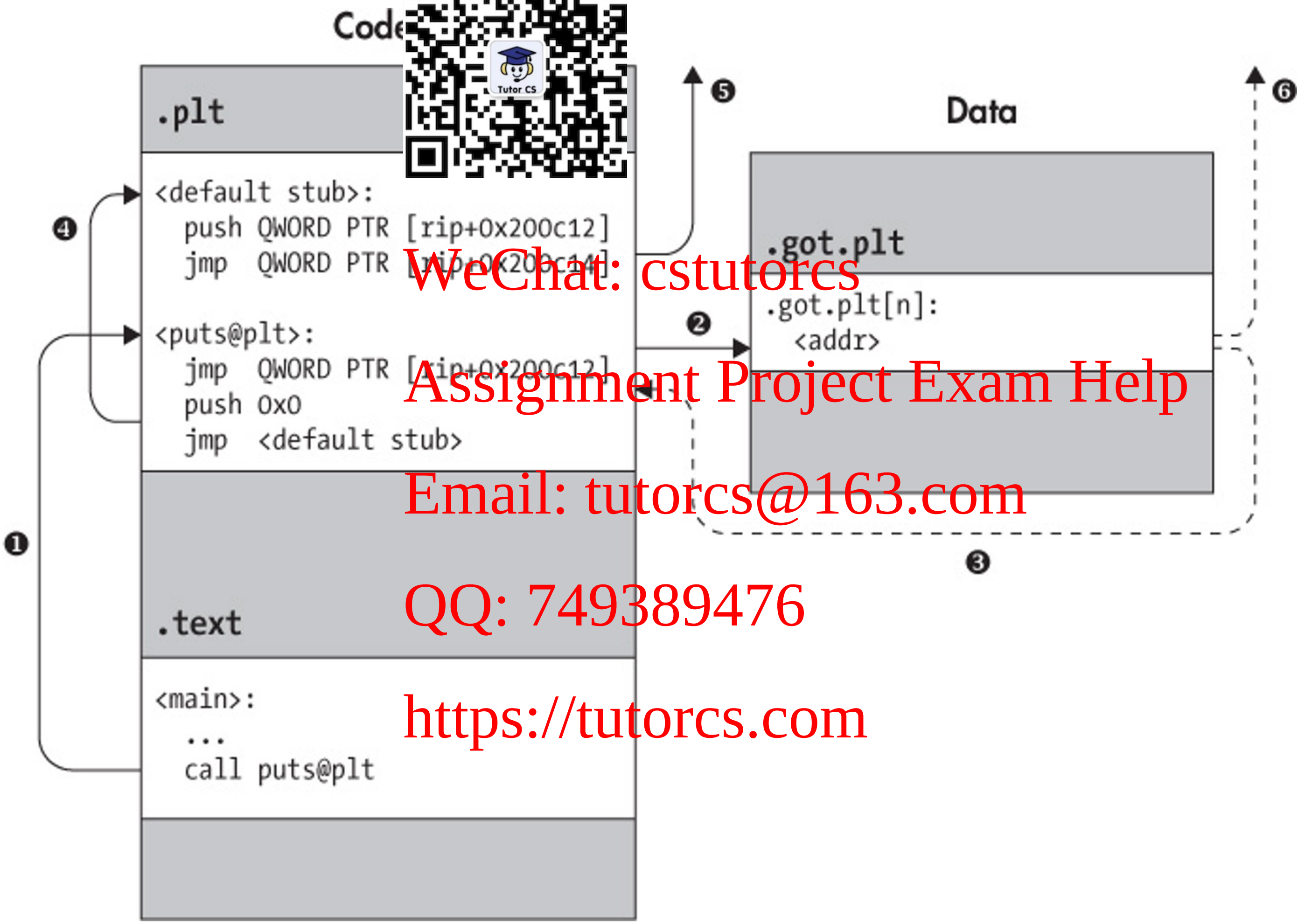
Sections - .bss, .data, and .rodata

- .rodata (read-only [illegible] edicated to storing constant values. Has type S[illegible]OGBITS.
- .data: default values of initialized variables. Writable. Has type SHT_PROGBITS.
- .bss (block started by symbol): reserve space for uninitialized variables. Has type SHT_NOBITS: doesn’t occupy bytes on disk. Writable.

Lazy Binding, .plt, and .got

- When a program is loaded into memory, functions from shared library are given dummy addresses – the process of resolving these to actual addresses are called relocation.
- Relocations are done when an unresolved symbol (denoting a function) is first referenced.
- This is called a 'lazy binding'. It is the default behaviour, but can be overridden, e.g. LD_BIND_NOW in Linux.
- Lazy binding requires two sections: Procedure Linkage Table (.plt) and Global Offset Table (.got / .got.plt).

Lazy Binding, .plt and .got



```
.plt
<default stub>:
  push QWORD PTR [rip+0x200c12]
  jmp  QWORD PTR [rip+0x200c14]

<puts@plt>:
  jmp  QWORD PTR [rip+0x200c12]
  push 0x0
  jmp  <default stub>

.text
<main>:
  ...
  call puts@plt
```

```
.got.plt
.got.plt[n]:
  <addr>
```

Lazy Binding, .plt, and .got

- One entry in .plt p[illegible] function. Each of the .plt entries has their own incr[illegible]D (see the result of **objdump -M intel –dj .plt <file>**)

- Each library has their own .plt and .got (.got.plt).

- Initially the entry in .got.plt refers back to the .plt for resolution.

- After resolution, the entry in .got.plt is patched with the address of the function.

Relocation Sections

- Indicated by SHT_[illegible] SHT_RELA
- Contains information used by the linker for performing relocations.
- Each SHT_RELA is a table of relocation entries: address where a relocation needs to be applied, instructions on how to resolve the value.

Relocation Sections (Example)

```
$ readelf --relocs a.out

Relocation section '.rela.dyn' at offset 0x380 contains 1 entries:
   Offset       Info           Type           Sym. Value    Sym. Name + Addend
0000600ff8 000300000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x398 contains 2 entries:
   Offset       Info           Type           Sym. Value    Sym. Name + Addend
0000601018 000100000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
0000601020 000200000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
```

Sections - .dynamic

- Useful during loading and setting up for execution.

- Contains a table of ELF64_Dyn structures, also referred to as tags.

- Also contains pointers to other important information required by the dynamic linker.

Sections - .init_array & .fini_array

- .init_array: array of pointers to functions to be used as constructors. What is the difference with .init?
- .fini_array: array of pointers to functions to be used as destructors.

Sections - .shstrtab, .symtab, .strtab, .dynsym, and .dynstr

- .shstrtab: array of [illegible]terminated strings that contain the names of all the sections in the binary. Indexed by section headers.
- .symtab: symbol table, each of which associates a symbolic name with a piece of code or data elsewhere in the binary. The actual strings containing the symbolic names are located in the .strtab section.
- .dynsym and .dynstr are analogous to .symtab and .strtab in the dynamic linking setting.

Program Headers

- Provides *a segme*[illegible]f the binary, as opposed to the section view provi[illegible]e section header table.
- Section view is for static linking purposes. Segment view is used by OS and dynamic linker when loading: locating relevant code and data, and decision on what to load to virtual memory.
- A segment consists of zero or more sections.
- It is used only in executable ELF.

Program Headers

PT_LOAD: intended to be loaded into memory when setting up the process. There are usually at least two – one for writable and one for non-writable sections.

PT_INTERP: contains the .interp section, which provides the name of the interpreter that is to be used to load the binary,

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

Program Headers

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

PT_DYNAMIC: contains the .dynamic section, which tells the interpreter how to parse and prepare the binary for execution.

PT_PHDR: encompasses the program header table.

Program Headers

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

p_offset: file offset at which the segment starts.

p_vaddr: virtual address at which it is to be loaded. For loadable segments, p_vaddr has to be the same as p_offset mod page size (which is typically 4,096 bytes).

p_filesz: file size of the segment

Program Headers

On some systems, it is possible to use the p_addr field to specify the physical memory to load the segment. On modern operating system such as Linux, this field is unused since they execute all binaries in virtual memory.

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executab
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

Program Headers

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

p_filesz: file size of the segment.
p_memsz: size of the segment in memory.
Mostly, p_filesz and p_memsz are the same, except for some cases where the sections only indicate the need for allocations, e.g., .bss.

Program Headers

p_flags: PF_X means segment is executable and is set for code segments. PF_W means segment is writable and is usually set for writable data segments, never for code segments. PF_R means segment is readable.

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

Program Headers

```
$ readelf --wide --segments a.out

 Elf file type is EXEC (Executable file)
 Entry point 0x400430
 There are 9 program headers, starting at offset 64

 Program Headers:
   Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
   PHDR           0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
   INTERP         0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R   0x1
       [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
   LOAD           0x000000 0x0000000000400000 0x0000000000400000 0x00070c 0x00070c R E 0x200000
   LOAD           0x000e10 0x0000000000600e10 0x0000000000600e10 0x000228 0x000230 RW  0x200000
   DYNAMIC        0x000e28 0x0000000000600e28 0x0000000000600e28 0x0001d0 0x0001d0 RW  0x8
   NOTE           0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R   0x4
   GNU_EH_FRAME   0x0005e4 0x00000000004005e4 0x00000000004005e4 0x000034 0x000034 R   0x4
   GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
   GNU_RELRO      0x000e10 0x0000000000600e10 0x0000000000600e10 0x0001f0 0x0001f0 R   0x1
...
```

p_align: indicates the required memory alignment. 0 or 1 means no alignment is required. Otherwise, it must be power of 2, and p_vaddr must be equal to p_offset, modulo p_align.

Program Headers

```
$ readelf --wide --segments a
...
Section to Segment mapping:
   Segment Sections...
    00
    01      .interp
    02      .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version
            .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata
            .eh_frame_hdr .eh_frame
    03      .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
    04      .dynamic
    05      .note.ABI-tag .note.gnu.build-id
    06      .eh_frame_hdr
    07
    08      .init_array .fini_array .jcr .dynamic .got
```

Resources

- Practical Binary Ana [illegible] ennis Andriesse. Chapter 2.
- https://people.redha [illegible] placek/src/devconf2012.pdf
- http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html

List of Commands Used

```
readelf -h <file>
```

Show the executable header of the file.

```
readelf --wide --sections <file>
```

Show the section headers in the file.

```
readelf --wide --segments <file>
```

Show the program headers in the file.

```
readelf --relocs <file>
```

Show the relocation symbols in the file.

List of Commands Used

readelf --symbols <

See the entries in the symbol table section.

readelf --dyn-syms <file>

See the entries in the dynamic symbol table section.

readelf -p <section, e.g., .shstrtab, .dynstr> <file>

Dump the content of a section in string format.

List of Commands Used

objdump -M intel -dj <section> <file>

Disassemble the content of a section and outputs the assembly code in Intel syntax. This is usually applied to sections that contain code, e.g., .text section or .plt section.

objdump -sj <section> <file>

Display the full content of a section (raw bytes). No disassembly is performed. This is suitable to display sections containing data, e.g., .rodata section or .got.plt section.