CS451 – Software Analysis

Lab Lecture

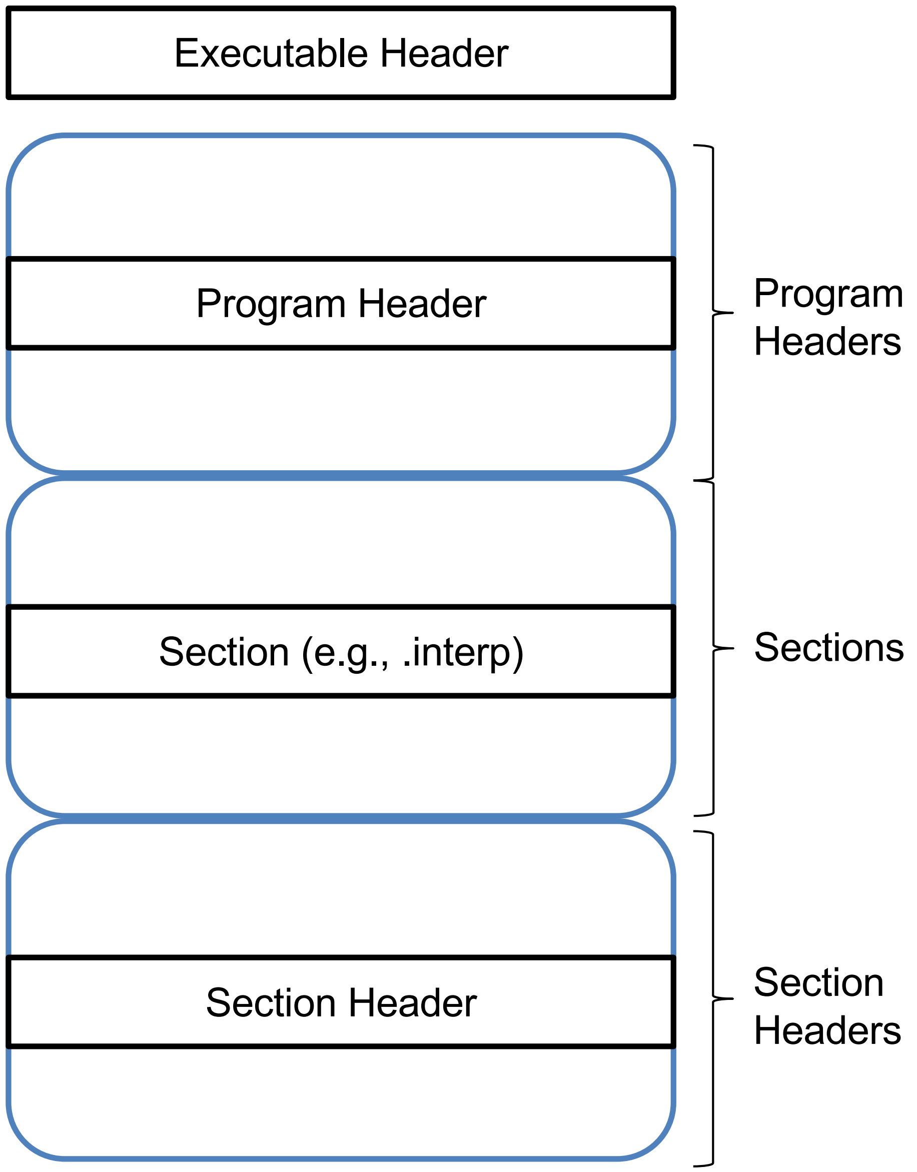
ELF

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

ELF

- Executable and Linkable Format
- Default format for Linux-based systems
- Used for executable files, object files, shared libraries and core dumps

ELF Format



- Division is used by linkers
- Sections contain data but handling each section is done through their section header
- The section headers can be found through the Executable Header

Executable Header

```
$ readelf -h ./first
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:               0x4004f0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          15712 (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:         30
  Section header string table index: 29
```

Definition of Executable Header

```
typedef struct
{
  unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
  Elf64_Half    e_type;          /* Object file type */
  Elf64_Half    e_machine;       /* Architecture */
  Elf64_Word    e_version;       /* Object file version */
  Elf64_Addr    e_entry;         /* Entry point virtual address */
  Elf64_Off     e_phoff;         /* Program header table file offset */
  Elf64_Off     e_shoff;         /* Section header table file offset */
  Elf64_Word    e_flags;         /* Processor-specific flags */
  Elf64_Half    e_ehsize;        /* ELF header size in bytes */
  Elf64_Half    e_phentsize;         /* Program header table entry size */
  Elf64_Half    e_phnum;         /* Program header table entry count */
  Elf64_Half    e_shentsize;         /* Section header table entry size */
  Elf64_Half    e_shnum;         /* Section header table entry count */
  Elf64_Half    e_shstrndx;      /* Section header string table index */
} Elf64_Ehdr;
```

Section Headers

```
elathan@sakura ~/epl451/labs/lab3 % readelf -SW ./test
There are 29 section headers, starting at offset 0x3928:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00000000000002a8 0002a8 00001c 00   A  0   0  1
  [ 2] .note.gnu.build-id NOTE           00000000000002c4 0002c4 000024 00   A  0   0  4
  [ 3] .note.ABI-tag     NOTE            00000000000002e8 0002e8 000020 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0000000000000308 000308 000024 00   A  5   0  8
…
```

Definition of a Section Header

```
typedef struct
{
  Elf64_Word    sh_name;        /* Section name (string tbl index) */
  Elf64_Word    sh_type;        /* Section type */
  Elf64_Xword   sh_flags;       /* Section flags */
  Elf64_Addr    sh_addr;        /* Section virtual addr at execution */
  Elf64_Off     sh_offset;      /* Section file offset */
  Elf64_Xword   sh_size;        /* Section size in bytes */
  Elf64_Word    sh_link;        /* Link to another section */
  Elf64_Word    sh_info;        /* Additional section information */
  Elf64_Xword   sh_addralign;   /* Section alignment */
  Elf64_Xword   sh_entsize;     /* Entry size if section holds table */
} Elf64_Shdr;
```

Section Header Fields

- `sh_name`
  - This is an index to a string table, which is hosted in a section called "`.shstrtab`"
  - The address of `.shstrtab` is defined in `e_shstrndx` (in th ELF header)
  - If the index is zero, it means that the section has no name
- sh_type
  - Every section has a type field (integer), related to the contents of the field

sh_type

```
/* Legal values for sh_type (section type).  */

#define SHT_NULL        0       /* Section header table entry unused */
#define SHT_PROGBITS    1       /* Program data */
#define SHT_SYMTAB      2       /* Symbol table */
#define SHT_STRTAB      3       /* String table */
#define SHT_RELA        4       /* Relocation entries with addends */
#define SHT_HASH        5       /* Symbol hash table */
#define SHT_DYNAMIC     6       /* Dynamic linking information */
#define SHT_NOTE        7       /* Notes */
#define SHT_NOBITS      8       /* Program space with no data (bss) */
#define SHT_REL         9       /* Relocation entries, no addends */
#define SHT_SHLIB       10      /* Reserved */
#define SHT_DYNSYM      11      /* Dynamic linker symbol table */
. . .
```

sh_flags

```
/* Legal values for sh_flags (section flags).  */

#define SHF_WRITE            (1 << 0)   /* Writable */
#define SHF_ALLOC            (1 << 1)   /* Occupies memory during execution */
#define SHF_EXECINSTR        (1 << 2)   /* Executable */
#define SHF_MERGE            (1 << 4)   /* Might be merged */
#define SHF_STRINGS          (1 << 5)   /* Contains nul-terminated strings */
#define SHF_INFO_LINK        (1 << 6)   /* `sh_info' contains SHT index */
#define SHF_LINK_ORDER       (1 << 7)   /* Preserve order after combining */
#define SHF_OS_NONCONFORMING (1 << 8)   /* Non-standard OS specific handling
                       required */
#define SHF_GROUP            (1 << 9)   /* Section is member of a group.  */
#define SHF_TLS              (1 << 10)  /* Section hold thread-local data.  */
#define SHF_COMPRESSED       (1 << 11)  /* Section with compressed data. */
#define SHF_MASKOS           0x0ff00000 /* OS-specific.  */
#define SHF_MASKPROC         0xf0000000 /* Processor-specific */
#define SHF_ORDERED          (1 << 30)  /* Special ordering requirement
                       (Solaris).  */
#define SHF_EXCLUDE          (1U << 31) /* Section is excluded unless
                       referenced or allocated (Solaris).*/
```

Other fields

- `sh_link`
  - A section may depend to another section (e.g., `.symtab` has pointers to the symbol names stored in `.strsymtab`)
- `sh_addr, sh_offset, sh_size`
  - Denote the virtual address the section will be mapped, the offset in the file, and the size of the section's payload
- `sh_info`
  - Additional information (depended to each section) for some sections
- `sh_addralign`
  - Some sections need to be aligned in a particular way (e.g., in an address that is multiple of 8 bytes)
- `sh_entsize`
  - The size of each record, for some sections that contain structured information (e.g., a table)

Sections

- There are some typical sections produced by common linkers
- Everybody can create their own sections with their own semantics
- It is common to have a first section with zero length, called the NULL section
  - This is an empty entry

Sections for Code and Data

- `.init/.fini`
  - They hold initialization and to be executed at exit code
- `.text`
  - The main code of the binary
- .bss, .data, .rodata
  - Data of the binary
- `.init_array, .fini_array`
  - Pointers for constructors and destructors

Sections for the Dynamic Loader

- `.rela.*, .rela`
  - Sections that contain relocation information used by the dynamic loader to resolve symbols at run-time
- `.dynamic`
  - Section that describes dependencies required for the dynamic loader

Sections for Symbols and Strings

- `.shstrtab, .strtab, .dynstr`
  - Tables that contain strings
- `.symtab, .dynsym`
  - Symbol table and dynamic symbol table

Program Headers

- Organizing the binary in sections produces a structure facilitated by linkers
- Executing a binary follows a different structure, which divides the binary in *segments*
- A segment, in principle, is a group of sections that is going to be mapped in the virtual address space

Program Headers

```
elathan@sakura ~/epl451/labs/lab3/elf % readelf --wide --segments ./test


Elf file type is DYN (Shared object file)
Entry point 0x1040
There are 11 program headers, starting at offset 64


Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  PHDR           0x000040 0x0000000000000040 0x0000000000000040 0x000268 0x000268 R   0x8
  INTERP         0x0002a8 0x00000000000002a8 0x00000000000002a8 0x00001c 0x00001c R   0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x000000 0x0000000000000000 0x0000000000000000 0x000530 0x000530 R   0x1000
  LOAD           0x001000 0x0000000000001000 0x0000000000001000 0x0001bd 0x0001bd R E 0x1000
  LOAD           0x002000 0x0000000000002000 0x0000000000002000 0x000170 0x000170 R   0x1000
  LOAD           0x002e18 0x0000000000003e18 0x0000000000003e18 0x000214 0x000218 RW  0x1000
  DYNAMIC        0x002e28 0x0000000000003e28 0x0000000000003e28 0x0001b0 0x0001b0 RW  0x8
  NOTE           0x0002c4 0x00000000000002c4 0x00000000000002c4 0x000044 0x000044 R   0x4
  GNU_EH_FRAME   0x002004 0x0000000000002004 0x0000000000002004 0x000044 0x000044 R   0x4
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
  GNU_RELRO      0x002e18 0x0000000000003e18 0x0000000000003e18 0x0001e8 0x0001e8 R   0x1
```

Segment View

```
 Section to Segment mapping:
  Segment Sections...
   00     
   01     .interp 
   02     .interp .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version 
.gnu.version_r .rela.dyn 
   03     .init .plt .plt.got .text .fini 
   04     .rodata .eh_frame_hdr .eh_frame 
   05     .init_array .fini_array .dynamic .got .got.plt .data .bss 
   06     .dynamic 
   07     .note.gnu.build-id .note.ABI-tag 
   08     .eh_frame_hdr 
   09     
   10     .init_array .fini_array .dynamic .got
```

Non-executable Objects

```
elathan@sakura ~/epl451/labs/lab3/elf % readelf --wide --segments ./test.o


There are no program headers in this file.
```