

ELF inside-out

Željko Vrba

November 6, 2007

Abstract

This tutorial lecture introduces the internals of the ELF format, which is used for executable and relocatable files, as well as for shared libraries. The following topics will be covered: ELF structure, dynamic loading, position-independent code, and linking. The lecture is UNIX oriented, but the basic concepts presented (sections, symbols, relocations, etc.) are applicable to all platforms. The `readelf` utility is introduced and its use exemplified on the i386 architecture. This lecture gives a basis for using other development tools (such as `objdump`, `ld` or `objcopy`), as well as for using more advanced tools (such as `ELFsh`).

ELF - Executable and Linking Format

- Used in some UNIX OS-es, most notably Solaris, *BSD and Linux
- Offers greater flexibility compared to COFF or a.out
- Used for executables, shared libraries (DLLs), relocatable objects
- Supports *position-independent code* (PIC), even for executables (PIE)
- Supports both 32- and 64-bit architectures; defines own fixed-width types (e.g. `Elf32_Word`, `Elf64_Off`...)
- Designed for easy loading (direct mapping).

Layout

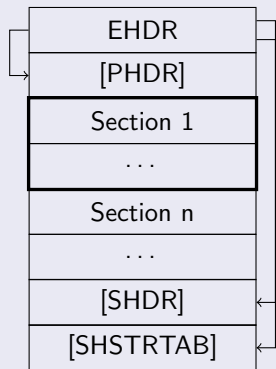
- Header
- Sections: contiguous sequences of bytes
- Segments: collection of sections

Data

- Symbols
- Relocations
- String table
- Hash table
- Debug information
- Dynamic linking information
- Program headers
- Code
- ...

Heavily cross-referenced:
compact, no redundancy (like DB
normal forms), a bit tricky to
process.
[OPTIONAL ITEM].

ELF layout



ELF header

```
zax ~/elf-itu2007 % readelf -h /bin/ls
```

ELF Header:

```
Magic:  7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class:                               ELF32
Data:                                 2's complement, little endian
Version:                              1 (current)
OS/ABI:                               UNIX - System V
ABI Version:                           0
Type:                                  EXEC (Executable file)
Machine:                               Intel 80386
Version:                               0x1
Entry point address:                   0x8049840
Start of program headers:               52 (bytes into file)
Start of section headers:               76104 (bytes into file)
Flags:                                  0x0
Size of this header:                    52 (bytes)
Size of program headers:                 32 (bytes)
Number of program headers:               9
Size of section headers:                 40 (bytes)
Number of section headers:               27
Section header string table index:      26
```

String table

- Just a section type, but ubiquitous
- Replace (variable-length) null-terminated string with an *index*
- Space conservation
- Other records become fixed-length: easier processing

Layout

a	◇	f	u	n	◇	
0	1	2	3	4	5	...

0 maps to string "a", 2 maps to string "fun", etc.

Section header

```
typedef struct
{
    Elf32_Word sh_name;
    Elf32_Word sh_type;
    Elf32_Word sh_flags;
    Elf32_Addr sh_addr;
    Elf32_Off sh_offset;
    Elf32_Word sh_size;
    Elf32_Word sh_link;
    Elf32_Word sh_info;
    Elf32_Word sh_addralign;
    Elf32_Word sh_entsize;
} Elf32_Shdr;
```

- Contiguous arrays of bytes in the file
- Linking and loading (segment parts)
- Virtual address (if alloc flag); file offset & size
- Type-dependent info

Sections

```
zax ~/elf-itu2007 % readelf -S /bin/ls
```

There are 27 section headers, starting at offset 0x12948:

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]		NULL	00000000	000000	000000	00		0	0	0
[1]	.interp	PROGBITS	08048154	000154	000013	00	A	0	0	1
[3]	.hash	HASH	08048188	000188	000310	04	A	4	0	4
[4]	.dynsym	DYNSYM	08048498	000498	000610	10	A	5	1	4
[5]	.dynstr	STRTAB	08048aa8	000aa8	000410	00	A	0	0	1
[8]	.rel.dyn	REL	0804900c	00100c	000028	08	A	4	0	4
[9]	.rel.plt	REL	08049034	001034	0002a0	08	A	4	11	4
[10]	.init	PROGBITS	080492d4	0012d4	000017	00	AX	0	0	4
[11]	.plt	PROGBITS	080492ec	0012ec	000550	04	AX	0	0	4
[12]	.text	PROGBITS	08049840	001840	00c9a8	00	AX	0	0	16
[13]	.fini	PROGBITS	080561e8	00e1e8	00001c	00	AX	0	0	4
[14]	.rodata	PROGBITS	08056220	00e220	003c68	00	A	0	0	32
[15]	.eh_frame_hdr	PROGBITS	08059e88	011e88	00002c	00	A	0	0	4
[16]	.eh_frame	PROGBITS	08059eb4	011eb4	00009c	00	A	0	0	4
[17]	.ctors	PROGBITS	0805a000	012000	000008	00	WA	0	0	4
[18]	.dtors	PROGBITS	0805a008	012008	000008	00	WA	0	0	4
[20]	.dynamic	DYNAMIC	0805a014	012014	0000d8	08	WA	5	0	4
[21]	.got	PROGBITS	0805a0ec	0120ec	000008	04	WA	0	0	4
[22]	.got.plt	PROGBITS	0805a0f4	0120f4	00015c	04	WA	0	0	4
[23]	.data	PROGBITS	0805a260	012260	00010c	00	WA	0	0	32
[24]	.bss	NOBITS	0805a380	01236c	00044c	00	WA	0	0	32
[26]	.shstrtab	STRTAB	00000000	012873	0000d5	00		0	0	1

Program header

```
typedef struct
{
    Elf32_Word p_type;
    Elf32_Off p_offset;
    Elf32_Addr p_vaddr;
    Elf32_Addr p_paddr;
    Elf32_Word p_filesz;
    Elf32_Word p_memsz;
    Elf32_Word p_flags;
    Elf32_Word p_align;
} Elf32_Phdr;
```

- Used only for EXE and SO
- Composed of sections
- Possible that memory size != file size (BSS)
- Note section (PT_NOTE) platform-specific

Segments

```
zax ~/elf-itu2007 % readelf -l /bin/ls
```

```
Elf file type is EXEC (Executable file)
```

```
Entry point 0x8049840
```

```
There are 9 program headers, starting at offset 52
```

```
Program Headers:
```

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x08048034	0x08048034	0x00120	0x00120	R E	0x4
INTERP	0x000154	0x08048154	0x08048154	0x00013	0x00013	R	0x1
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x08048000	0x08048000	0x11f50	0x11f50	R E	0x1000
LOAD	0x012000	0x0805a000	0x0805a000	0x0036c	0x007cc	RW	0x1000
DYNAMIC	0x012014	0x0805a014	0x0805a014	0x000d8	0x000d8	RW	0x4
NOTE	0x000168	0x08048168	0x08048168	0x00020	0x00020	R	0x4
GNU_EH_FRAME	0x011e88	0x08059e88	0x08059e88	0x0002c	0x0002c	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x4
PAX_FLAGS	0x000000	0x00000000	0x00000000	0x00000	0x00000		0x4

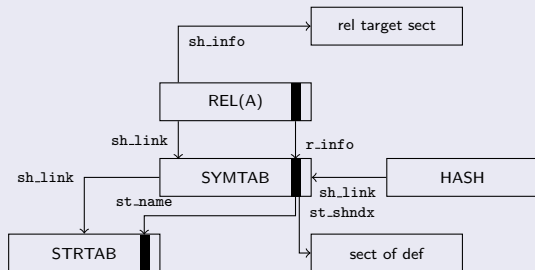
```
Section to Segment mapping:
```

```
Segment Sections...
```

```
00
01 .interp
02 .interp .note.ABI-tag .hash .dynsym .dynstr [...etc]
03 .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
04 .dynamic
05 .note.ABI-tag
06 .eh_frame_hdr
07
```

- Pieces: object files, static and dynamic libraries
- Linking: putting the pieces together
- Determine final addresses, fixup references: symbols and relocations
- “Linkers and Loaders”: <http://www.iecc.com/linker/>

Data cross-referencing



Example

C code

```
#include <stdlib.h>

int x;
static int y;

int f(void)
{
    return abs(x+y)*random();
}

int main(void)
{
    return 0;
}
```

Disassembly (main not shown)

```
zax ~/elf-itu2007 % objdump -dr ex.o
ex.o:      file format elf32-i386
Disassembly of section .text:

00000000 <f>:
0:  55                push   %ebp
1:  89 e5             mov    %esp,%ebp
3:  53                push   %ebx
4:  83 ec 04         sub   $0x4,%esp
7:  8b 15 00 00 00 00  mov   0x0,%edx
9:  R_386_32        x
d:  a1 00 00 00 00    mov   0x0,%eax
e:  R_386_32        .bss
12: 8d 04 02         lea   (%edx,%eax,1),%eax
15: 99                cld
16: 89 d3             mov   %edx,%ebx
18: 31 c3             xor   %eax,%ebx
1a: 29 d3             sub   %edx,%ebx
1c: e8 fc ff ff ff   call  1d <f+0x1d>
1d: R_386_PC32      random
21: 0f af c3         imul  %ebx,%eax
24: 83 c4 04         add   $0x4,%esp
27: 5b                pop   %ebx
28: 5d                pop   %ebp
29: c3                ret
```

Symbols and Relocations

Symbols

```
zax ~/elf-itu2007 % readelf -s ex.o
```

```
Symbol table '.symtab' contains 12 entries:
```

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FILE	LOCAL	DEFAULT	ABS	ex.c
2:	00000000	0	SECTION	LOCAL	DEFAULT	1	
3:	00000000	0	SECTION	LOCAL	DEFAULT	3	
4:	00000000	0	SECTION	LOCAL	DEFAULT	4	
5:	00000000	4	OBJECT	LOCAL	DEFAULT	4	y
6:	00000000	0	SECTION	LOCAL	DEFAULT	6	
7:	00000000	0	SECTION	LOCAL	DEFAULT	5	
8:	00000000	42	FUNC	GLOBAL	DEFAULT	1	f
9:	00000004	4	OBJECT	GLOBAL	DEFAULT	COM	x
10:	00000000	0	NOTYPE	GLOBAL	DEFAULT	UND	random
11:	0000002a	25	FUNC	GLOBAL	DEFAULT	1	main

Relocations

```
zax ~/elf-itu2007 % readelf -r ex.o
```

```
Relocation section '.rel.text' at offset 0x34c contains 3 entries:
```

Offset	Info	Type	Sym.Value	Sym.Name
00000009	00000901	R_386_32	00000004	x
0000000e	00000401	R_386_32	00000000	.bss
0000001d	00000a02	R_386_PC32	00000000	random

Symbol structure

```
typedef struct
{
    Elf32_Word    st_name;
    Elf32_Addr    st_value;
    Elf32_Word    st_size;
    unsigned char st_info;
    unsigned char st_other;
    Elf32_Section st_shndx;
} Elf32_Sym;
```

- `st_info`: type, binding (local, global, weak)
- `st_other`: visibility (default, internal, hidden, protected)

Relocation structure

```
typedef struct
{
    Elf32_Addr r_offset;
    Elf32_Word r_info;
} Elf32_Rel;
```

- `Elf32_Rela`: additional `r_addend` member
- `r_offset`: where to apply relocation to
- `r_info`: relocation type & symbol index

Example: after linking

```
gcc ex.c -o ex
```

Disassembly (main not shown)

```
08048398 <f>:
8048398: 55          push   %ebp
8048399: 89 e5      mov   %esp,%ebp
804839b: 53        push   %ebx
804839c: 83 ec 04   sub   $0x4,%esp
804839f: 8b 15 a4 95 04 08  mov  0x80495a4,%edx
80483a5: a1 a0 95 04 08  mov  0x80495a0,%eax
80483aa: 8d 04 02   lea   (%edx,%eax,1),%eax
80483ad: 99        cld
80483ae: 89 d3      mov   %edx,%ebx
80483b0: 31 c3     xor   %eax,%ebx
80483b2: 29 d3     sub   %edx,%ebx
80483b4: e8 0f ff ff ff  call  80482c8 <random@plt>
80483b9: 0f af c3   imul %ebx,%eax
80483bc: 83 c4 04   add   $0x4,%esp
80483bf: 5b       pop   %ebx
80483c0: 5d       pop   %ebp
80483c1: c3       ret
```

- Process of preparing the executable for running.
- PT_LOAD sections are mapped in memory to their vaddrs.
- The kernel sets up the stack (cmdline pointer vector, environ pointer vector, auxv, cmdline strings, environ strings)

Static executables

- PT_INTERP segment is absent.
- Control transferred to the entry point.

Dynamic executables

- PT_INTERP segment is present; contains the interpreter name.
- The program interpreter is mapped.
- Control is transferred to the interpreter's entry point.
- The interpreter (dynamic linker) resolves library dependencies, maps them as necessary and starts the program.

Environment variables

- LD_PRELOAD: function overrides (SUID/SGID: special treatment)
- LD_LIBRARY_PATH: as PATH, but for libs
- LD_TRACE_LOADED_OBJECTS: used by ldd
- LD_BIND_NOW: resolve symbols immediately
- many others: see `man ld.so`

The auxiliary vector

- Generic type-value mapping
- Terminated by AT_NULL

Auxv structure

```
typedef struct
{
    uint32_t a_type;
    union
    {
        uint32_t a_val;
        /* ptr removed
        for 64-bit compat */
    } a_un;
} Elf32_auxv_t;
```

Tags required for dyn linker (linux)

```
#define AT_PHDR    3 /* Program headers for program (pointer) */
#define AT_PHENT   4 /* Size of program header entry */
#define AT_PHNUM   5 /* Number of program headers */
#define AT_PAGESZ  6 /* System page size */
#define AT_ENTRY   9 /* Entry point of program */
```

Position-independent code

- Code that works independent of loading address.
- Absolute references not allowed.
- Compiled with -fPIC flag; data references through GOT.

Disasm

```
00000000 <f>:
0: 55          push  %ebp
1: 89 e5       mov   %esp,%ebp
3: 56          push  %esi
4: 53          push  %ebx
5: e8 fc ff ff call  6 <f+0x6>
6: R_386_PC32 __i686.get_pc_thunk.bx
a: 81 c3 02 00 00 00 add   $0x2,%ebx
c: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
10: 8b 83 00 00 00 00 mov   0x0(%ebx),%eax
12: R_386_GOT32 x
16: 8b 10       mov   (%eax),%edx
18: 8b 83 00 00 00 00 mov   0x0(%ebx),%eax
1a: R_386_GOTOFF .bss
```

Disasm (cont'd)

```
1e: 8d 04 02    lea   (%edx,%eax,1),%eax
21: 99          cld
22: 89 d6       mov   %edx,%esi
24: 31 c6       xor   %eax,%esi
26: 29 d6       sub   %edx,%esi
28: e8 fc ff ff call  29 <f+0x29>
29: R_386_PLT32 random
2d: 0f af c6    imul %esi,%eax
30: 5b         pop   %ebx
31: 5e         pop   %esi
32: 5d         pop   %ebp
33: c3         ret
```

Getting the IP: x86_32

```
0804843d <_i686.get_pc_thunk.bx>:  
804843d: 8b 1c 24 mov (%esp),%ebx  
8048440: c3      ret
```

Relocation types

- $R_386_32 = S+A$
- $R_386_PC32 = S+A-P$
- $R_386_GOTPC = GOT+A-P$
- $R_386_GOTOFF = S+A-GOT$
- $R_386_PLT32 = L+A-P$
- $R_386_GOT32 = G+A$

Getting the IP: x86_64

- `%rip` as base register
- No need to call the thunk

Abbreviations

- S: value of symbol in rel.
- A: relocation addend
- P: place being relocated (`r_offset`)
- L: place in PLT
- GOT: GOT address
- G: offset into GOT

```
zax ~/elf-itu2007 % readelf -S ex
[Nr] Name           Type           Addr      Off      Size    ES Flg Lk Inf Al
[17] .got             PROGBITS      000016b0 0006b0 000010 04  WA  0  0  4
[18] .got.plt         PROGBITS      000016c0 0006c0 000018 04  WA  0  0  4
```

- Table that indirect references to global data
- Two parts: data and PLT
- Defined by symbol `_GLOBAL_OFFSET_TABLE_` (on x86); may be in the “middle” of `.got` section (allows for negative offsets)
- `GOT[0]`: address of the dynamic structure (`_DYNAMIC` symbol)

[EXAMPLE]

Absolute PLT

```

080492ec .PLT0:
80492ec: ff 35 f8 a0 05 08  pushl  0x805a0f8  [GOT+4]
80492f2: ff 25 fc a0 05 08  jmp    *0x805a0fc  [GOT+8]
80492f8: 00 00                add    %al,(%eax)  [NOPs]

080492fc <readlink@plt>: [.PLT1]
80492fc: ff 25 00 a1 05 08  jmp    *0x805a100  [name1_in_GOT]
8049302: 68 00 00 00 00    push  $0x0         [$offset]
8049307: e9 e0 ff ff ff    jmp    80492ec     [.PLT0@PC]

```

PIC PLT

```

0000043c .PLT0:
43c: ff b3 04 00 00 00  pushl  0x4(%ebx)   [GOT+4]
442: ff a3 08 00 00 00  jmp    *0x8(%ebx) [GOT+8]
448: 00 00                add    %al,(%eax) [NOPs]

0000044c <random@plt>: [.PLT1]
44c: ff a3 0c 00 00 00  jmp    *0xc(%ebx) [*name1@GOT(%ebx)]
452: 68 00 00 00 00    push  $0x0         [$offset]
457: e9 e0 ff ff ff    jmp    43c         [.PLT0@PC]

```

PLT: function resolution

- 1 Upon program loading: initialize GOT[1] and GOT[2]
- 2 If PLT is PIC: %ebx *must* be initialized to correct GOT (each SO has own PLT/GOT)
- 3 Calling function (eg. random) will transfer control to its PLT slot
- 4 The 1st insn jumps to GOT entry (initially holding the address of the following pushl)
- 5 The offset of R_386_JMP_SLOT relocation entry is pushed onto stack. (rel offset = GOT offset table entry)
- 6 The code jumps to 0th PLT slot which pushes GOT[1] and jumps to GOT[2], transferring control to dynamic linker.
- 7 Dynamic linker unwinds the stack, resolves the real symbol address and places it in its GOT entry.
- 8 Further calls directly invoke the target function.

The dynamic section

```
zax ~/elf-itu2007 % readelf -d ex
```

```
Dynamic section at offset 0x5f0 contains 20 entries:
```

Tag	Type	Name/Value
0x00000001	(NEEDED)	Shared library: [libc.so.6]
0x0000000c	(INIT)	0x424
0x0000000d	(FINI)	0x5bc
0x00000004	(HASH)	0xd4
0x00000005	(STRTAB)	0x2ec
0x00000006	(SYMTAB)	0x17c
0x0000000a	(STRSZ)	144 (bytes)
0x0000000b	(SYMENT)	16 (bytes)
0x00000003	(PLTGOT)	0x16c0
0x00000002	(PLTRELSZ)	24 (bytes)
0x00000014	(PLTREL)	REL
0x00000017	(JMPREL)	0x40c
0x00000011	(REL)	0x3dc
0x00000012	(RELSZ)	48 (bytes)
0x00000013	(RELENT)	8 (bytes)
0x6ffffffe	(VERNEED)	0x3ac
0x6fffffff	(VERNEEDNUM)	1
0x6ffffff0	(VERSYM)	0x37c
0x6ffffffa	(RELCOUNT)	2
0x00000000	(NULL)	0x0

Another tag: DT_RPATH / DT_RUNPATH: library search path.

```
gcc -fPIC -shared -Wl,-rpath,/opt ex.c -o ex
```

Topics not covered

- PIE, SO dependencies, hash table
- AMD64 and other architectures
- Many fields in the dynamic section
- (De)initialization process
- Windows and PIC (or its lack thereof)
- Debug information (a *huge* topic!)

- ELF specification:
`http://www.x86.org/ftp/manuals/tools/elf.pdf`
- AMD64 ABI: `http://www.x86-64.org/documentation/abi.pdf`
- `/usr/include/elf.h`
- `http://people.redhat.com/drepper/dsohowto.pdf`