ELF
Executable and Linkable Format

Mikołaj Baczyński

Plan

1.Komenda readelf
2. O ELF’ie
3.Typy plików
4.Nagłówek ELF
5.Segmenty
6.Typy nagłówków programu
7.Sekcje
8.Rodzaje Sekcji
9.Symbole
10.Realokacja
11.Linkowanie dynamiczne/statyczne

readelf

readelf – jedno z najbardziej użytecznych narzędzi do analizy plików binarnych ELF. Przetwarza każdy bit, by wydobyć informacje o obiekcie, by móc potem wykonać inżynierie wsteczną (reverse engineering).

- readelf -S <obiekt> - pobiera nagłówek sekcji
- readelf -l <obiekt> - pobiera nagłówek programu
- readelf -h <obiekt> - pobiera nagłówek ELF
- readelf -s <obiekt> - pobiera tablice symboli

O ELF’ie

Standardowy format binarny dla systemów Unix i Unix flavors(Sun’s Solaris,Mac OS X, IBM’s AIX).

Używany jest do plików:

- Executables – wykonywalnych
- shared libraries – bibliotek współdzielonych
- object files – relokowalnych
- coredump files – zrzuty pamięci

Typy plików

ET_REL – plik relokowalne,często z rozszerzeniem *.o. Plik, który nie został jeszcze zlinkowany (z innym plikiem ET_REL) do pliku wykonywalnego, współdzielonego, lub innego pliku relokowalnego. Nie posiada segmentów. Możemy uzyskać go kompilując, bez linkowania: *gcc -c plik*

ET_EXEC – plik wykonywalny, nazywany również programem, jest punktem wejścia(entry point) jak proces zaczyna działać. Zawiera binarną reprezentację instrukcji dla procesora. Powstaje po skompilowaniu i zlinkowaniu.

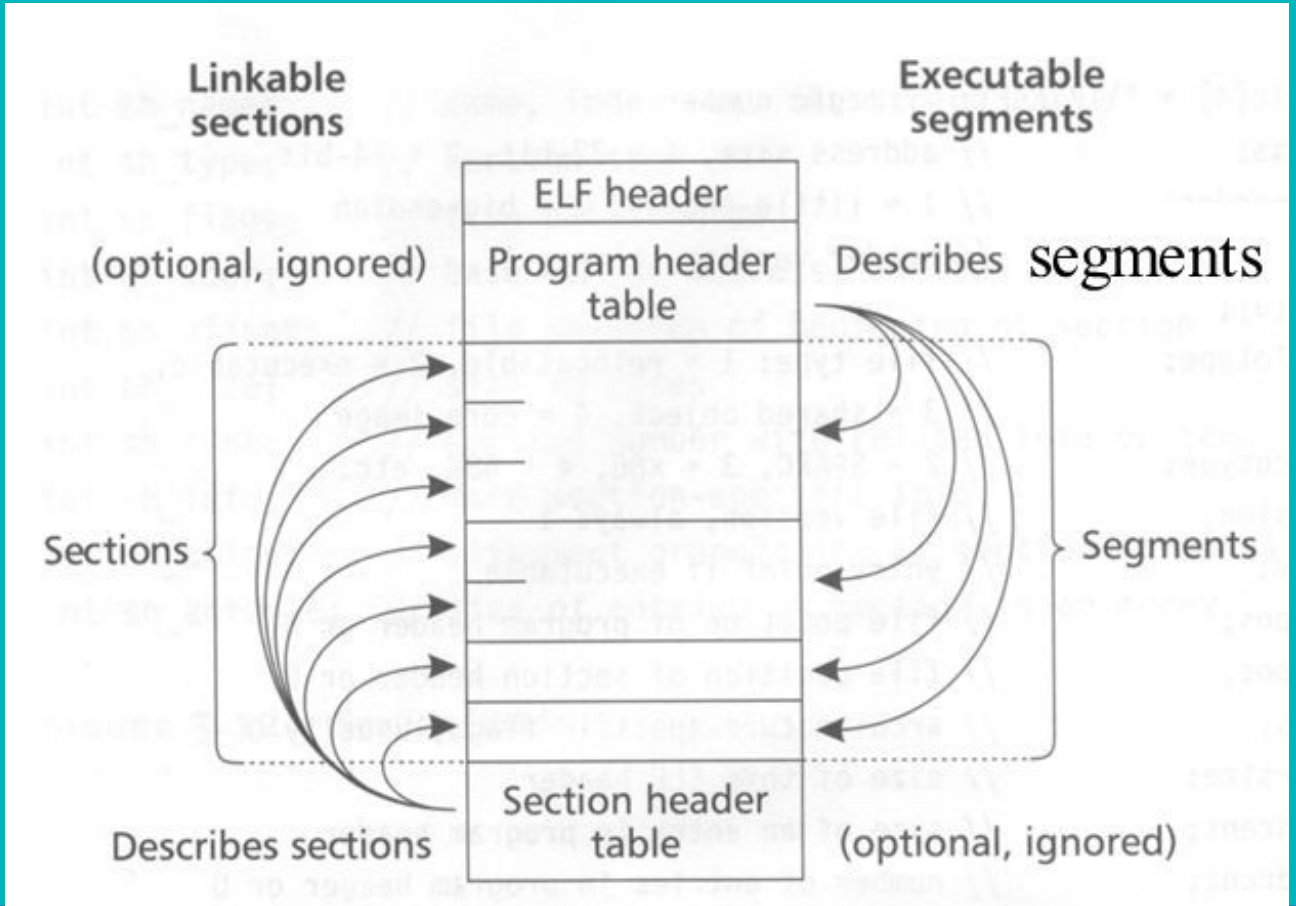
Typy plików

ET_DYN – plik (biblioteka) współdzielona – dynamicznie linkowany obiekt. Jest linkowany i ładowany w runtime'ie. Aby otrzymać taki plik, musimy kompilować z flagami
-fPIC i -shared.

ET_CORE – core dump – plik zawiera zrzut mapy pamięci, powstaje, gdy program się nieoczekiwanie zakończył, lub proces wysłał sygał naruszenia pamięci. Te pliki możemy czytać za pomocą GDB, by zobaczyć co spowodowało „wysypanie" się programu

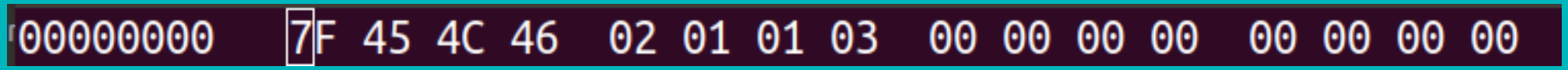
ET_NONE – plik nieznany, lub jeszcze nie zdefiniowany.

Struktura


Linkable sections
Executable segments
ELF header
(optional, ignored)
Program header table
Describes segments
Sections
Segments
Section header table
Describes sections
(optional, ignored)

Nagłówek ELF

Jest to pierwsze 16 bajtów pliku – zwane Magic. Możemy je wyświetlić za pomocą narzędzia *hexedit plik.*



Pierwsze 4 definiują, że jest to ELF: 7F – prefix, 45=E, 4c=L, 46=F (ASCII)
5 bajt – architektura 01/02 – 34/64 bitowa
6 bajt – 01/02 - little-endian/big-endian

Kolejne bajty mówią o systemie operacyjnym, typie pliku, adresie pamięci w którym program zaczyna działanie i offsetach do segmentów, sekcji, ich ilości i wielkości.

Nagłówek ELF

Narzędzie readelf -h zinterpretuje za nas każdy bajt nagłówka i wypisze w bardziej zrozumiałym języku.

```
mikolaj@mikolaj-Lenovo-G50-70:~$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 03 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - GNU
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x400a30
  Start of program headers:          64 (bytes into file)
  Start of section headers:          842592 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         6
  Size of section headers:           64 (bytes)
  Number of section headers:         33
  Section header string table index: 32
```

Segmenty

Segmenty opisywane są przez nagłówki programu.

Są interpretowane przez Kernel w czasie ładowania programu i opisują layout pamięci (w którym są dane, lub kod wykonywalny) pliku wykonywalnego na dysku.

Segmenty są opisywane przez struktury Elf32/64_Phdr, które są rozmieszczone w sposób ciągły, czyli występują w pamięci jeden po drugim.

Flagi segmentów

Każdy segment może mieć następujące flagi:
R - segment do odczytu
E – segment wykonywalny
W – segment do zapisu

Typy segmentów

PT_LOAD – plik wykonywalny ma przynajmniej ten jeden typ segmentu. Ten segment zostnaie załadowany, lub zmapowany do pamięci.

Na przykład plik ELF dynamicznie linkowany będzie zawierał 2 segmenty tego typu:
1 – segment tekstowy dla kodu programu
2 – segment danych dla zmiennych globalnych i informacji o dynamicznym linkowaniu

Typy segmentów

PT_DYNAMIC – posiadają je pliku które są linkowane dynamicznie – zawiera inforamcję konieczne do linkowania:
-info o wejściach realokacji
-lista biblliotek współdzielonych potrzebnych do zlinkowania

PT_INTERP – zawiera tylko rozmiar i stringa z lokalizacją gdzie jest interpreter programu (przeważnie jest to lokalizacja dynamicznego linkera)

GNU_STACK – przechowuje info o stosie

Typy segmentów

Segmenty możemy wyświetlić komendą: *readelf -l plik*

```
mikolaj@mikolaj-Lenovo-G50-70:~$ readelf -l a.out

Elf file type is EXEC (Executable file)
Entry point 0x400a30
There are 6 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000b591e 0x000000000000b591e  R E    0x200000
  LOAD           0x000000000000b6120 0x000000000006b6120 0x000000000006b6120
                 0x00000000000051b8 0x00000000000068e0  RW     0x200000
  NOTE           0x0000000000000190 0x0000000000400190 0x0000000000400190
                 0x0000000000000044 0x0000000000000044  R      0x4
  TLS            0x000000000000b6120 0x000000000006b6120 0x000000000006b6120
                 0x0000000000000020 0x0000000000000060  R      0x8
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x000000000000b6120 0x000000000006b6120 0x000000000006b6120
                 0x0000000000002ee0 0x0000000000002ee0  R      0x1
```

Sekcje

Wewnątrz każdego segmentu dane, lub kod jest podzielony na sekcje. Sekcja zawiera tablice nagłówków sekcji która ma referencje do lokalizacji i wielkości tych sekcji.
Jest ona szczególnie potrzebna do linkowania i debugowania.

Program może wykonać się bez section table, ponieważ ona nie definiuje layoutu pamięci (robi to tablica nagłówków programu) .

Jeśli nagłówki sekcji są „zaginione”, nie oznacza to że nie ma sekcji. Po prostu nie ma na nie referencji

Rodzaje sekcji

.text – zawiera kod programu. Jest on read-only więc ma typ PROGBITS.

.rodata – zawiera dane read-only takie jak string z linii C printf(„Hello World!”);

.data – zawiera zainicjalizowane zmienne globalne.

.bss – zawiera niezainicjalizowane zmienne. Zmienne są inicjalizowane w czasie ładowania programu na 0 i mogą dostać konkretną wartość w czasie wykonania programu. Sekcja jest NOBITS, ponieważ aktualnie nie ma danych.

Sekcje

readelf -l pokazuje, które sekcje są mapowane na segmenty, co pozwala zobaczyć relacje pomiędzy sekcją i segmentem.

readelf -S wyświetla wszystkie nagłówki sekcji, oraz ich wielkość, typ itp..

Symbole

Symbole są symbolicznymi referencjami do danych, lub kodu. (zmienne, albo funkcje).

np. printf() ma symbol wejścia, który wskazuje na dynamiczną tablice symboli .dynsym.

.dynsym zawiera symbole pochodzące z dodatkowych źródeł np. libc

.symtab zawiera to co dynsym, ale też zmienne i funkcje lokalne z naszego kodu.

Realokacja

Proces, który łączy referencje symboli z ich definicjami. W realokacji są specjalne rekordy z metadanymi opisującymi jak upakować instrukcje np. przy linkowaniu.

Przykład.

Linkowanie statyczne

Kiedyś wszystko był linkowane statycznie. Kiedy program używał dodatkowych funkcji, cała biblioteka był kompilowana do pliku wykonywalneo, czyli wszystkie definicje (normalnie będące w bibliotekach) są przechowywane w pliku.

Jeśli chcemy skompilować kod statycznie musimy dodać do *gcc* opcje *-static*.

Linkowanie dynamiczne

Jakie kroki wykonuje linker:

1) Obsługuje własne realokacje.
2) Szuka w DYNAMIC wszystkich tagow DT_NEEDED, które zawierają wskaźnik na nazwy lokalizacji potrzebnych bibliotek.
3) Kiedy je znajdzie mapuje je do pamięci.
4) Biblioteki dodają swoje tablice symboli
5) Linker tworzy liste zbudowaną ze struktur link_map
6) Jak linker skończy budować listę zależności , zajmuje się realokacją każdej biblioteki.

```
struct link_map
  {
    ElfW(Addr) l_addr; /* Base address shared object is loaded at.  */
    char *l_name;      /* Absolute file name object was found in.  */
    ElfW(Dyn) *l_ld;   /* Dynamic section of the shared object.  */
    struct link_map *l_next, *l_prev; /* Chain of loaded objects.  */
  };
```

Literatura

- Learning Linux Binary Analysing – Ryan O’Neill
- blog.k3170makan.com – blog
- man7.org/linux/man-pages/man5/elf.5.html
- https://pl.wikipedia.org/wiki/Executable_and_Linkable_Format

KONIEC
Dziękuje za uwagę

Mikołaj Baczyński