# Estructura de un Archivo ELF: Una Guía Detallada

Un archivo ELF (Executable and Linkable Format) es el formato estándar para ejecutables, bibliotecas compartidas, archivos de objetos y volcados de memoria ( core dumps ) en sistemas Unix-like (como Linux). Es una estructura altamente organizada que permite al sistema operativo cargar y ejecutar programas, o al enlazador combinar diferentes piezas de código.

Un archivo ELF se compone de varias partes principales:

# 1. Cabecera ELF (ELF Header)

Es lo primero en cualquier archivo ELF y describe la estructura general del archivo. Es como la **tarjeta de identificación** del archivo. Contiene campos cruciales para interpretar el resto del archivo.

#### Información clave para ft\_nm:

- Magic Number (e\_ident): Los primeros 4 bytes son 0x7F 'E' 'L' 'F', que identifican el archivo como ELF. iYa lo estás comprobando!
- Clase de Arquitectura (e\_ident[EI\_CLASS]): Indica si el ELF es de 32 bits (ELFCLASS32) o 64 bits (ELFCLASS64). iYa lo estás detectando con find bits!
- Endianness (e\_ident[EI\_DATA]): Especifica el orden de bytes (little-endian o big-endian). iYa lo estás detectando con find\_endianness! Es crucial para leer correctamente valores de múltiples bytes (direcciones, offsets, tamaños).
- Versión ELF ( e\_ident[EI\_VERSION] ): La versión del formato ELF.
- Tipo de Objeto ( e type ):
  - ET REL: Archivo reubicable (objeto, .o).
  - ET EXEC: Archivo ejecutable.
  - ET DYN: Biblioteca compartida ( .so ) o ejecutable cargado dinámicamente.
  - ET CORE: Volcado de memoria.
- Arquitectura de la Máquina (e\_machine): Indica la arquitectura del procesador (ej. EM\_X86\_64 para x86-64, EM\_386 para i386, EM\_AARCH64 para ARM64).
- Versión ( e version ): Debe ser EV CURRENT .
- **Dirección de Entrada (** e\_entry ): Para ejecutables, es la dirección de memoria virtual donde el sistema operativo debe comenzar la ejecución.
- Offset de la Tabla de Cabeceras de Programa ( e\_phoff ): La posición dentro del archivo donde comienza la Tabla de Cabeceras de Programa.
- Offset de la Tabla de Cabeceras de Sección ( e\_shoff ): La posición dentro del archivo donde comienza la Tabla de Cabeceras de Sección.

- Flags ( e flags ): Flags específicos de la arquitectura.
- Tamaño de la Cabecera ELF ( e ehsize ): Tamaño de esta cabecera en bytes.
- Tamaño de las Entradas de la Cabecera de Programa (e\_phentsize): Tamaño de una entrada en la Tabla de Cabeceras de Programa.
- Número de Entradas de la Cabecera de Programa (e\_phnum): Número de entradas en la Tabla de Cabeceras de Programa.
- Tamaño de las Entradas de la Cabecera de Sección ( e\_shentsize ): Tamaño de una entrada en la Tabla de Cabeceras de Sección.
- Número de Entradas de la Cabecera de Sección ( e\_shnum ): Número de entradas en la Tabla de Cabeceras de Sección.
- Índice de la Sección de Nombres (e\_shstrndx): El índice de la entrada en la Tabla de Cabeceras de Sección que contiene los nombres de las secciones.

# 2. Tabla de Cabeceras de Programa (Program Header Table o Segment Header Table)

Esta tabla es crucial para el **cargador de programas** del sistema operativo. Describe cómo el archivo debe ser mapeado en la memoria en tiempo de ejecución. Cada entrada en esta tabla se llama **Program Header** o **Segment Descriptor**.

**Información clave para** ft\_nm: (Aunque nm se enfoca más en las secciones, entender los segmentos es útil para el contexto).

- **Tipo de Segmento (** p\_type ): Indica el propósito del segmento (ej., PT\_LOAD para segmentos cargables, PT DYNAMIC para información de enlace dinámico).
- Offset de Archivo ( p offset ): Offset del segmento dentro del archivo ELF.
- **Dirección Virtual (** p vaddr ): Dirección virtual donde se cargará el segmento en memoria.
- Tamaño en Archivo (p filesz): Tamaño del segmento en el archivo.
- Tamaño en Memoria (p\_memsz): Tamaño del segmento en memoria (puede ser mayor que
  p\_filesz si incluye datos inicializados a cero).
- Flags ( p\_flags ): Permisos de lectura/escritura/ejecución ( PF\_R , PF\_W , PF\_X ).
- Alineación (p align): Requisito de alineación del segmento en memoria.

# 3. Tabla de Cabeceras de Sección (Section Header Table)

Esta tabla describe las secciones del archivo. Las **secciones** son unidades más pequeñas y lógicas dentro de los segmentos, utilizadas por el enlazador y las herramientas de depuración. Aquí es donde nm busca la mayoría de su información.

#### Información clave para ft nm:

Cada entrada en esta tabla es un **Section Header** y contiene:

- Nombre de la Sección (sh\_name): Un offset al string que contiene el nombre de la sección en la sección de strings de los nombres de sección (.shstrtab).
  - Necesitas encontrar la sección .shstrtab para resolver estos nombres.
- **Tipo de Sección (sh\_type)**: El propósito de la sección (ej., SHT\_PROGBITS para código o datos, SHT\_SYMTAB para la tabla de símbolos, SHT\_STRTAB para la tabla de strings).
- Flags (sh\_flags): Atributos de la sección (ej., SHF\_ALLOC si se carga en memoria, SHF EXECINSTR si contiene código ejecutable, SHF WRITE si es escribible).
- Dirección Virtual (sh addr): Dirección virtual de la sección si se carga en memoria.
- Offset de Archivo (sh offset): La posición de la sección dentro del archivo ELF.
- Tamaño de la Sección (sh size): El tamaño de la sección en bytes.
- Enlace (sh\_link): Índice de otra sección relacionada (ej., para una tabla de símbolos, esto puede apuntar a su tabla de strings asociada).
- Información (sh\_info): Información adicional específica del tipo de sección.
- Alineación (sh addralign): Requisito de alineación de la sección en memoria.
- Tamaño de Entrada (sh\_entsize): Si la sección contiene entradas de tamaño fijo (como la tabla de símbolos), este es el tamaño de cada entrada.

# 4. Secciones Importantes para ft\_nm

Dentro de la Tabla de Cabeceras de Sección, hay algunas secciones específicas que nm necesita analizar:

- symtab (Symbol Table Section):
  - Contiene la tabla de símbolos del archivo. Cada entrada es una estructura Elf32\_Sym o
     Elf64 Sym.
  - Necesitas:
    - Su sh offset para saber dónde empieza en el archivo.
    - Su sh size para saber cuántos bytes ocupa.
    - Su sh entsize para saber el tamaño de cada entrada de símbolo.
    - Su sh link para encontrar la tabla de strings de símbolos (.strtab) asociada.
- .strtab (String Table Section):
  - Contiene las cadenas de caracteres (nombres) de los símbolos listados en .symtab . Los símbolos en .symtab solo almacenan un offset a esta sección para su nombre.

#### Necesitas:

- Su sh offset para saber dónde empieza.
- Su sh size para saber su tamaño.
- .shstrtab (Section Header String Table Section):
  - Contiene los nombres de todas las secciones del archivo. La cabecera ELF (e\_shstrndx)
     apunta a esta sección.

#### Necesitas:

- Su sh offset para saber dónde empieza.
- Su sh size para saber su tamaño.

## 5. Entradas de la Tabla de Símbolos (Symbol Table Entry)

Cada entrada en .symtab es un **símbolo**. Aquí es donde obtendrás la información que nm imprime.

#### Información clave para ft nm:

Para cada Elf32 Sym o Elf64 Sym (la estructura depende de si el ELF es de 32 o 64 bits):

- st\_name: Un offset a la cadena de caracteres que es el nombre del símbolo, dentro de la sección .strtab asociada.
- st\_value : El valor del símbolo. Puede ser una dirección, un tamaño, o cualquier otra cosa dependiendo del contexto. Para la mayoría de los símbolos, es la dirección del símbolo.
- st size: El tamaño del símbolo.
- st info: Contiene el tipo y el binding del símbolo.
  - ELF32\_ST\_BIND(st\_info) / ELF64\_ST\_BIND(st\_info) : Extrae el binding(ej., STB\_LOCAL, STB\_GLOBAL, STB\_WEAK).
  - ELF32\_ST\_TYPE(st\_info) / ELF64\_ST\_TYPE(st\_info): Extrae el tipo(ej., STT NOTYPE, STT OBJECT, STT FUNC, STT SECTION, STT FILE).
- st other: Usado para visibilidad (ej., STV DEFAULT, STV HIDDEN).
- st\_shndx: El índice de la sección a la que se refiere el símbolo. Por ejemplo, si es una función, este es el índice de la sección de código donde reside. Puede ser un valor especial como SHN\_ABS (valor absoluto), SHN\_COMMON (símbolo común) o SHN\_UNDEF (símbolo indefinido, es decir, externo).

# Guía Clara para Construir tu ft\_nm (Flujo de Información)

Basado en lo que ya tienes y la información del ELF, aquí está la secuencia lógica para construir tu nm:

#### 1. Lectura del Archivo y Mapeo en Memoria (iYa lo haces!):

- Abre el archivo.
- o Obtén su tamaño (fstat).
- Mapea el archivo en memoria (mmap).
- o Verifica la firma ELF ( 0x7F 'E' 'L' 'F' ).
- Detecta la clase (32/64 bits) y el endianness. (iYa lo tienes!)

#### 2. Parsear la Cabecera ELF (Elf32 Ehdr o Elf64 Ehdr):

- Dependiendo de aux->bits, castear aux->file\_content\_ptr a Elf32\_Ehdr \* o Elf64 Ehdr \* .Almacenar esto en aux->elf32 header o aux->elf64 header.
- Esta cabecera te dará los offsets y números de entradas para las Tablas de Cabeceras de Programa y Sección.
- **Crucial:** El e\_shoff (offset a la tabla de cabeceras de sección) y e\_shnum (número de secciones) serán vitales. También e shstrndx (índice de la sección .shstrtab).

#### 3. Localizar la Tabla de Cabeceras de Sección:

- Usa e\_shoff de la cabecera ELF para encontrar el inicio de la Tabla de Cabeceras de Sección en la memoria mapeada.
- Necesitarás un puntero a Elf32\_Shdr o Elf64\_Shdr (dependiendo de aux->bits ).

#### 4. Localizar la Sección de Nombres de Sección (.shstrtab):

- Usa el índice e\_shstrndx de la Cabecera ELF para acceder a la entrada correspondiente en la Tabla de Cabeceras de Sección.
- De esa entrada (la que describe .shstrtab ), obtén su sh offset y sh size.
- Esto te dará la dirección y tamaño de la sección que contiene los nombres de todas las demás secciones. Esto es esencial para nombrar las secciones.

#### 5. Iterar a través de la Tabla de Cabeceras de Sección:

- Recorre las e shnum entradas de la Tabla de Cabeceras de Sección.
- o Para cada entrada, necesitas:
  - Obtener su nombre usando sh\_name como offset en la sección .shstrtab que acabas de localizar.

- Buscar secciones con tipo SHT\_SYMTAB y SHT\_STRTAB (para la tabla de símbolos y su tabla de strings asociada, respectivamente).
- Registrar sus offsets, tamaños y enlaces (sh link).

#### 6. Localizar la Tabla de Símbolos ( .symtab ) y su Tabla de Strings ( .strtab ):

- Una vez que has iterado por las cabeceras de sección, identifica la sección SHT\_SYMTAB.
   Esta es tu tabla de símbolos.
- Usa su campo sh\_link para encontrar el índice de la sección SHT\_STRTAB asociada (su tabla de strings de símbolos).
- o Obtén los offsets y tamaños de estas dos secciones.

#### 7. Parsear las Entradas de la Tabla de Símbolos:

- Usa el sh\_offset de .symtab y el sh\_entsize de .symtab para iterar sobre cada Elf32 Sym o Elf64 Sym en la memoria mapeada.
- Para cada símbolo:
  - Obtén su nombre usando st\_name como offset en la sección .strtab que acabas de localizar.
  - Extrae el **tipo de símbolo** y el **binding** de st\_info (usando las macros ELF32\_ST\_TYPE, ELF32\_ST\_BIND o sus equivalentes para 64 bits).
  - Obtén su **valor** de st\_value.
  - Identifica su **sección** usando st shndx.

#### 8. Determinar el Tipo de Símbolo (el carácter que muestra nm ):

- Esta es la parte más compleja. El carácter (T, t, D, d, B, b, U, W, etc.) depende de una combinación de:
  - st shndx: Si es shn undef (U), shn abs (A), shn common (C).
  - st\_type: Sies STT\_FUNC (Tot), STT\_OBJECT (D, d, B, b), STT\_SECTION (Sos).
  - st\_bind: Si es STB\_GLOBAL (mayúscula) o STB\_LOCAL (minúscula), STB WEAK (W o w).
  - Flags de la sección referenciada: Si la sección es escribible (SHF\_WRITE), ejecutable (SHF\_EXECINSTR), etc. Por ejemplo, .text es T, .data es D, .bss es B.
  - Visibilidad ( st other ): Afecta si un símbolo débil es W o w.

#### 9. Filtrar y Ordenar Símbolos (para las flags de nm ):

- o Aplica las reglas de las flags que soportes ( −a , −g , −u , −r , −p ).
- o Ordena los símbolos. Por defecto, nm los ordena alfabéticamente. La flag -r los invierte.

## 10. **Imprimir la Salida:**

o Formatea la salida según las convenciones de nm (dirección, tipo de símbolo, nombre).