# Linux incident response - ELF files

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-11-04

## Understanding Linux ELF Files in Incident Response and Investigations

Linux ELF (Executable and Linkable Format) files are fundamental to the Linux operating system, serving as executable binaries and shared libraries. When it comes to incident response and digital investigations, comprehending the intricacies of ELF files is paramount. In this exploration, we will dissect ELF files, both on disk and in memory, focusing on elements critical for incident response professionals.

### ELF File Format:

ELF files consist of a header and multiple sections, each serving a specific purpose. The header, represented by e_ident, contains essential information about the file, including its class (32-bit or 64-bit), data encoding, file version, and OS/ABI (Operating System/ Application Binary Interface). The class field distinguishes between 32-bit and 64-bit architectures. For 32-bit ELF files, the class is ELFCLASS32, while for 64-bit files, it is ELFCLASS64.

The file signature identifies ELF files, with the 32-bit signature being **7F 45 4C 46** and the 64-bit signature being **7F 45 4C 46 02 01 01** in hexadecimal representation. In ASCII it is "**.ELF"** and "**.ELF**..." respectively.

### Key Elements in ELF Header:

**e_type**: Specifies the file type (executable, shared object, etc.), aiding in determining the file's purpose.

**e_entry**: Indicates the virtual address where the program execution starts, providing insight into the entry point of the code.

**e_phoff, e_phentsize, e_phnum**: Describes the program header table's offset, entry size, and the number of entries. Program headers contain information about segments, vital for loading the file into memory.

**e_shoff, e_shentsize, e_shnum**: Specifies the section header table's offset, entry size, and the number of entries. Section headers contain details about various sections in the ELF file, such as code, data, and symbols.

**e_shstrndx**: Represents the index of the section header containing section names, facilitating easy identification of sections.

### In-Memory Structures:

When an ELF file is loaded into memory, sections like .text, .data, and .bss contain critical information. .text holds executable code, .data stores initialized data, and .bss contains

uninitialized data. During incident response, examining these sections in memory aids in understanding the program's behaviour and any potentially malicious activities.

## ELF File Metadata

ELF files contain various metadata that provide essential information about the file's structure, attributes, and execution requirements. Some of the key metadata stored in an ELF file include:

1. **ELF Header (, , , etc.):**
   : ELF identification information, specifying the file class (32-bit or 64-bit), data encoding, OS/ABI, and ELF version.: Indicates the file type, such as executable, shared object, or relocatable.: Specifies the architecture for which the binary is intended (e.g., x86, ARM, MIPS).: ELF version number.: Virtual address where the program execution starts.
2. **Program Header Table (Segments):**
   Describes the segments of the ELF file, specifying how the binary should be loaded into memory. Each entry includes information about the segment's type, file offset, virtual address, physical address, size in file, and size in memory.
3. **Section Header Table:**
   Contains information about various sections within the ELF file, such as code, data, symbols, and string tables. Each entry in the section header table provides details about the section's type, virtual address, file offset, size, and other attributes.
4. **Sections (, , , etc.):**
   Store executable code, initialized data, uninitialized data, and other specific types of information. Sections hold the actual content and instructions of the program.
5. **Symbol Table and String Table Sections:**
   The symbol table () contains information about symbols used in the binary, including function and variable names, their types, and addresses. The string table () holds the strings associated with these symbols, making them human-readable.
6. **Dynamic Section:**
   Contains dynamic linking information, including shared library dependencies, relocation entries, and symbol resolutions during runtime.
7. **Relocation Sections:**
   Store the relocation information needed to adjust addresses in the ELF file when it is loaded at a different memory location.
8. **Dynamic String Table and Dynamic Symbol Table:**
   Similar to the string table and symbol table but specific to dynamic linking. They store strings and symbols used in the dynamic linking process.
9. **Version Sections:**
   Contain versioning information for symbols, allowing proper resolution of symbols during dynamic linking when multiple versions of a library are available.

**Windows vs Linux note**

The main distinction between a PE (Portable Executable) file header and an ELF (Executable and Linkable Format) header lies in their structure and content. In a PE file, the header includes a DOS header, a PE signature, and a COFF (Common Object File Format) header, containing essential information like file architecture, entry point, and section details. Additionally, PE files have specific sections for imports, exports, resources, exceptions, and certificates. In contrast, an ELF header encompasses identification information (), specifying file class, data encoding, and OS/ABI, along with the file type () and target architecture (). The ELF header leads to program and section header tables, which detail segments and sections within the file, allowing for dynamic linking, symbol resolution, and runtime execution.

## Malware, File Packing, and Unpacking Techniques

In the realm of cybersecurity, malware authors consistently devise strategies to bypass detection, with file packing emerging as a prevalent evasion technique. File packing involves compressing or encrypting an ELF file, rendering its content opaque to traditional inspection methods. This cloak of invisibility is achieved through algorithms that compress the file's size or scramble its content, making it challenging for security tools to discern malicious intent. Detecting and understanding packed ELF files is pivotal for incident responders, as it unveils a layer of complexity in malware analysis.

### How File Packing Works:

File packing operates by compressing the executable, reducing its size, or encrypting its content to obfuscate its true nature. This process often involves using encryption algorithms or compression techniques like LZ77 or Huffman coding. When an ELF file is packed, it becomes a self-contained archive, encapsulating the original binary along with decompression routines. As the malware executes, the decompression routines dynamically restore the original code and data into memory. This dynamic unpacking hampers static analysis methods, requiring a more sophisticated approach to dissect the malware's behaviour.

### The Role of UPX:

One prominent tool employed for file packing is the Ultimate Packer for eXecutables (UPX). UPX is a widely used open-source software packer that applies various compression algorithms to ELF files, transforming them into highly compressed executables. UPX not only reduces the file size but also complicates reverse engineering attempts by altering the file's structure. Security professionals encounter UPX-packed files frequently due to UPX's popularity among malware authors seeking to conceal their creations.

### Recognizing and Unpacking Packed ELF Files:

Detecting packed ELF files requires astute observation. Security tools can often flag packed files based on irregularities in their headers, such as discrepancies in section sizes or unexpected compression markers. Unusual entropy levels in the binary can also hint at packing. To unpack these files, incident responders employ dynamic analysis techniques, executing the malware in a controlled environment, monitoring its behaviour, and capturing the unpacked content during runtime. Moreover, specialized tools and scripts are used to identify and neutralize compression routines, revealing the pristine, unpacked code and aiding investigators in understanding the malware's true functionality.

In essence, comprehending file packing techniques and recognizing tools like UPX are crucial for incident responders. By mastering the art of identifying and unpacking these concealed executables, security professionals can delve deeper into the analysis, revealing the malware's underlying mechanics. This expertise empowers responders to formulate effective countermeasures, fortifying digital landscapes against evolving threats and ensuring a resilient defence against sophisticated cyber adversaries.

---

## About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

**Web:** www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858