IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 1, January 2018

An Anatomy of Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Formats for Digital Forensic Analysis and Anti-Virus Design Purposes

Mohammed A. Saleh

Computer Science Department, College of Sciences and Arts in Ar Rass, Qassim University, Kingdom of Saudi Arabia

Abstract: The era of forensic analysis and anti-virus design requires a clear anatomy of Windows Executable File (EXE) and Linux Executable Linkable Format File (ELF) Formats, especially for beginners to these fields; thus, this research comes out. First, this research identifies data structures for both files formats, namely EXE and ELF files formats. After that, it classifies them according to headers, sections, and resources based on specific features and functionalities. Finally, this research proposes analysis guidelines for EXE and ELF files formats forensic analysis and anti-virus design purposes.

Keywords: Anatomy of EXE format file, anatomy of ELF format file, anti-virus design, digital forensic analysis.

I. INTRODUCTION

Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) files formats dominate executable files in computer era. Windows Executable File (EXE) in an executable file for Windows operating system, while Linux Executable and Linkable Format File (ELF) is an executable file for Unix-Like operating systems, such as Linux and BSD. Since Windows and Linux operating systems are the most widely used operating systems on the globe, EXE and ELF are commonly spread, too [1][2].
However, digital forensic analyst and anti-virus designer seek to be familiar with EXE and ELF file formats in order to analyse them properly, and adopt their solutions accordingly. Digital forensic analyst needs to identify file format and architecture, and then investigate it. After that, in case the malicious executable file is detected, an anti-virus needs to update its database, so it be able to detect it later on. The following sections analyse and discuss an anatomy of Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) formats for digital forensic analysis and anti-virus design purposes in granular details [2][3][4][5][6].

II. LITERATURE REVIEW

In reality, Windows Executable File (EXE), and Linux Executable and Linkable Format File (ELF) structures have many identical features and functionalities, and on the other hand, they have not. Both of EXE and ELF format's files are formed and constructed throughout data structures. In general, ELF format's file comprises of three main successive structures, namely Elfxx_Ehdr structure, Elfxx_Phdr structure, and Elfxx_Shdr structure, where xx is 32 for 32-bits structure, or 64 for 64-bits structure. On the other hand, EXE format's file encompasses of five main consecutive structures, such as IMAGE_DOS_HEADER structure, IMAGE_NT_HEADERS structure, IMAGE_FILE_HEADER structure, IMAGE_OPTIONAL_HEADER structure, and IMAGE_SECTION_HEADER structure. In addition, IMAGE_OPTIONAL_HEADER structure contains sub structure called DataDirectory used to include further sub structures like IMAGE_EXPORT_DIRECTORY structure, IMAGE_IMPORT_DESCRIPTOR structure, and IMAGE_RESOURCE_DATA_ENTRY structure. The following Table 1 summarizes structures of EXE and ELF format's files [7][8][4][9][10][11][12][13][14].

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 1, January 2018

TABLE 1: A Summary of Windows Executable File (EXE) Data Structures, and Linux Executable and Linkable Format File (ELF) Data Structures

ELF File Format	EXE File Format
• **Elf32_Ehdr Struct:** e_ident (0x7f ELF) e_type (reloc, exe, …) e_machine (Arch.) e_entry (virtAddr of .text) e_phoff (ph table's offset) e_shoff (sh table's offset) e_ehsize e_phentsize e_phnum e_shentsize e_shnum • **Elf32_Phdr Struct:** p_type (LOAD, DYNAMIC, INTERP) p_offset (beginning of file) p_vaddr p_paddr p_filesz p_memsz p_flags (X, W, R) p_align • **Elf32_Shdr Struct:** sh_name (.data, .text, .bss) sh_type (PROGBITS, SYMTAB, NOBITS, REL) sh_flags (Data_w, Code_X) sh_addr sh_size sh_offset (beginning of file) sh_addralign sh_entsize	• **IMAGE_DOS_HEADER Struct:** e_magic (MZ) e_ss e_sp e_ip e_cs e_lfanew (PE file header Offset) **MS-DOS 2.0 Stub Program** • **IMAGE_NT_HEADERS Struct:** Signature (PE00) FileHeader OptionalHeader • **IMAGE_FILE_HEADER Struct:** Machine (Arch.) NumberOfSections TimeDateStamp SizeOfOptionalHeader acteristics (exe, dll, system files, ...) • **IMAGE_OPTIONAL_HEADER Struct:** Magic (0x10b=PE32, 0x20b=PE32+) SizeOfCode (Size of executable code) SizeOfInitializedData (Size of Initialized Data) SizeOfUninitializedData (Size of Uninitialized Data) AddressOfEntryPoint (virtAddr of .text) BaseOfCode (Relative offset of code) BaseOfData (Relative offset of Data) ImageBase SectionAlignment FileAlignment SizeOfImage SizeOfHeaders NumberOfRvaAndSizes DataDirectory[NumberOfRvaAndSizes] • **IMAGE_SECTION_HEADER Struct:** Name VirtualAddress SizeOfRawData PointerToRawData acteristics (Code, Initialized data, Uninitialized data, Resource, +X, +W, +R, …) **7. DataDirectory[NumberOfRvaAndSizes]** • **IMAGE_EXPORT_DIRECTORY Struct:** Name Base NumberOfFunctions NumberOfNames AddressOfFunctions AddressOfNames AddressOfNameOrdinals • **IMAGE_IMPORT_DESCRIPTOR Struct:**

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 1, January 2018

	OriginalFirstThunk TimeDateStamp ForwarderChain Name FirstThunk • **IMAGE_IMPORT_BY_NAME Struct:** Hint Name[1] • **IMAGE_RESOURCE_DIRECTORY Struct**: NumberOfNamedEntries NumberOfIdEntries • **IMAGE_RESOURCE_DIRECTORY_ENTRY Struct:** NameId Data • **IMAGE_RESOURCE_DATA_ENTRY Struct:** Data Size CodePage Reserved

III. Classifying Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Data Structures

The following Table 2 presents a classification of identical headers for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) data structures based on a feature/functionality. It shows a Feature/Functionality in the first column, ELF Format File in the second column, and EXE Format File in the third column. These data structures are used to identify file format, type, and architecture. In addition, they are used to calculate offsets to sections, determine program entry point, and set sections permissions [4][7][8][4][9][10][11][12][13][14].

TABLE 2: A Classification of Identical headers for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Data Structures based on a Feature/Functionality

Feature/Functionality	**ELF Format File**	**EXE Format File**
Magic Value	• **Elf32_Ehdr Struct:** o e_ident (**0x7f ELF**)	• **IMAGE_DOS_HEADER Struct:** o e_magic (**MZ**)
Next Structs Offset	• **Elf32_Ehdr Struct:** o e_phoff o e_shoff	• **IMAGE_DOS_HEADER Struct:** o e_lfanew (Offset to PE) • **IMAGE_NT_HEADERS Struct:** o Signature (PE00) o FileHeader o OptionalHeader
Machine Architecture	• **Elf32_Ehdr Struct:** o e_machine	• **IMAGE_FILE_HEADER Struct:** o Machine
File Type	• **Elf32_Ehdr Struct:**	• **IMAGE_FILE_HEADER Struct:** o acteristics (exe, dll, system files, ...)

	o e_type (reloc, exe, …)	
Program Entry Point	• **Elf32_Ehdr Struct:** o e_entry (.text)	• **IMAGE_OPTIONAL_HEADER Struct:** o AddressOfEntryPoint (.text)
Program Sections Number	• **Elf32_Ehdr Struct:** o e_shnum	• **IMAGE_FILE_HEADER Struct:** o NumberOfSections
Permissions	• **Elf32_Phdr Struct:** o p_flags (X, W, R) • **Elf32_Shdr Struct:** o sh_flags (Data_w, Code_X)	• **IMAGE_SECTION_HEADER Struct:** o acteristics (Code, Initialized data, Uninitialized data, Resource, +X, +W, +R, …)

As well, Table 3 below presents a classification of identical sections for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) data structures based on a feature/functionality. The purposes of these data structures are used to locate and load text section, namely .text section, and execute it accordingly, which represents executable instructions. Besides that, they locate and load an initialized and uninitialized sections, such as .data, .rodata, .pdata, and .bss sections that are used for storing global and local variables, strings names, and constants. In addition, they locate and load a resource section, namely .rsrc that is used only for Windows Executable File (EXE). Table 4 below provides granular details of .rsrc section [4][7][8][4][9][10][11][12][13][14].

TABLE 3: A Classification of Identical Sections for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Data Structures based on a Feature/Functionality

Feature/Functionality	**ELF Format File**	**EXE Format File**
Text Section (.text)	• **Elf32_Shdr Struct:** o sh_name (.text) o sh_type (PROGBITS, REL) o sh_flags (Code_X) o sh_addr o sh_size o sh_offset (beginning of file)	• **IMAGE_OPTIONAL_HEADER Struct:** o SizeOfCode (Size of executable code) o BaseOfCode (Relative offset of code) • **IMAGE_SECTION_HEADER Struct:** o Name o VirtualAddress o SizeOfRawData o PointerToRawData o acteristics (Code, +X, +W, +R, …)
Initialized Data Section (.data, .rodata, .pdata, idata, edata)	• **Elf32_Shdr Struct:** o sh_name (.data) o sh_type (PROGBITS) o sh_flags (Data_w) o sh_addr o sh_size o sh_offset (beginning of file)	• **IMAGE_OPTIONAL_HEADER Struct:** o SizeOfInitializedData (Size of Initialized Data) o BaseOfData (Relative offset of Data) • **IMAGE_SECTION_HEADER Struct:** o Name o VirtualAddress o SizeOfRawData o PointerToRawData

		o acteristics (Initialized data, +X, +W, +R, …)
Uninitialized Data Section (.bss)	• **Elf32_Shdr Struct:** o sh_name (.bss) o sh_type (PROGBITS) o sh_flags (Data_w) o sh_addr o sh_size o sh_offset (beginning of file)	• **IMAGE_OPTIONAL_HEADER Struct:** o SizeOfUninitializedData (Size of Uninitialized Data) • **IMAGE_SECTION_HEADER Struct:** o Name o VirtualAddress o SizeOfRawData o PointerToRawData o acteristics (Uninitialized data, Resource, +X, +W, +R, …)
Resource Section (.rsrc)	• **N/A**	• **IMAGE_OPTIONAL_HEADER Struct:** o NumberOfRvaAndSize o DataDirectory[NumberOfRvaAndSizes] • **IMAGE_SECTION_HEADER Struct:** o Name o VirtualAddress o SizeOfRawData o PointerToRawData o acteristics (Resource, +X, +W, +R, …)
Section Packer, Encryptor, and Protector	• **Manipulate sections, and change program entry point.**	• **Manipulate sections, and change program entry point.**

Finally, Table 4 below details a classification of a resource section for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) data structures based on a feature/functionality. The data structures identify a resource type whether is imported functions, exported functions, or merely a resource like buttons, menus, etc. After that, these sections are loaded accordingly [4][7][8][4][9][10][11][12][13][14].

TABLE 4: A Classification of Resource Section for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Data Structure based on a Feature/Functionality

Feature/Functionality	ELF Format File	EXE Format File
Resource Section (.rsrc)	• **N/A**	• **IMAGE_OPTIONAL_HEADER Struct:** o DataDirectory[NumberOfRvaAndSizes] • **IMAGE_EXPORT_DIRECTORY Struct:** o Name o Base o NumberOfFunctions o NumberOfNames o AddressOfFunctions o AddressOfNames o AddressOfNameOrdinals

		• **IMAGE_IMPORT_DESCRIPTOR Struct:** o OriginalFirstThunk o TimeDateStamp o ForwarderChain o Name o FirstThunk • **IMAGE_IMPORT_BY_NAME Struct:** o Hint o Name[1] • **IMAGE_RESOURCE_DIRECTORY Struct**: o NumberOfNamedEntries o NumberOfIdEntries • **IMAGE_RESOURCE_DIRECTORY_ENTRY Struct:** o NameId o Data • **IMAGE_RESOURCE_DATA_ENTRY Struct:** o Data o Size o CodePage o Reserved

IV. Malware Analysis Guidelines for Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) Formats Forensic Analysis and Anti-Virus Design Purposes

1. Firstly, inspect headers values of EXE and ELF formats file, as explained in Table 2 above, in order to determine the right structures values. However, some malware authors obfuscate the written malware by appending or padding various bytes in begin, specific locations, or end of malware sample in order to prevent malware analysis and make it harder to the experts. Hence, a further deeper inspection is required to carve the correct structure values from malware sample.
2. After that, inspect existence of sections header values as depicted in Table 3 above. Here there are two dissimilar options:
   i. The sections names are identical to names shown in Table 3 above, then follow step 3 below.
   ii. The sections names are different from names shown Table 3 above, and then follow step 4 below.
3. Investigate and analyze **.text** section, which has the executable instructions, using static and dynamic analysis. Additionally, analyze **.data**, **.rodata**, **.pdata, idata**, **edata**, and other data sections, which hold variables' names and values, strings' names and values, constants' names and values, imported functions, and exported functions. Indeed, this step leads to explain malwares harms and negative impacts.
4. In this case, the default sections' names of a malware sample are changed by malware authors themselves to obfuscate the analysis and make it harder for the experts. They accomplish this mission by using a Packer, an Encryptor, or a Protector to change the original value of **Program Entry Point** to a new one. During program execution, it returns the original sections names and executes them accordingly. Therefore, the malware sample needs to be decrypted or unpacked first, and then follow step iii above.
5. Afterwards, inspect and investigate any extra sections names other than whose mentioned above, which created and adopted by some malware authors, by following step iii above.
6. Then, inspect and investigate **.rsrc** section of exe format file, which hold an important data to malware sample. Usually, malware authors hide harmful instructions inside this section.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified

Vol. 7, Issue 1, January 2018

7. Finally, a forensic analyst writes down a detailed forensic analysis based on a malware sample investigation, especially headers data structures, executable instructions and data sections, and packing and encrypting of sections data structures. As well, an anti-virus designer computes digest hash values, such as sha256 hash function, for a malware sample and it's all sections to update its anti-virus database, hence an anti-virus detects a malware sample later on.

V. CONCLUSION

Digital forensic analysis and an anti-virus design are important services and tools in computer era, which need a deep understanding of the most common executable files, namely Windows Executable File (EXE) and Linux Executable Linkable Format File (ELF) files formats. Thus, this research studied, analysed, and discussed an anatomy of Windows Executable File (EXE) and Linux Executable and Linkable Format File (ELF) formats for digital forensic analysis and anti-virus design purposes. As well, it proposed analysis guidelines for EXE and ELF files formats forensic analysis and anti-virus design purposes. The proposed guidelines demonstrate sequential steps to be followed by digital forensic analyst and an anti-virus designer in order to analyse and classify any executable files whether is a malicious or benign file. Overall, these guidelines involve identify data structures for both files formats, and then classify them according to headers, sections, and resources based on specific features and functionalities. Finally, a forensic analyst writes down a detailed forensic analysis, and an anti-virus designer computes digest hash values and update its anti-virus database, therefore it detects a malware sample later on. In future work, we plan to deign detectable methods based on different approaches for detecting malicious EXE and ELF executable files.

REFERENCES

[1] L. Wu, "Automatic Detection Model of Malware Signature for Anti-virus Cloud Computing," *10th IEEE/ACIS Int. Conf. Comput. Inf. Sci. Autom.*, pp. 1–3, 2011.
[2] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless Sandboxes : Evading Malware Analysis Systems using Wear-and-Tear Artifacts," *IEEE Symp. Secur. Priv. Spotless*, pp. 1009–1024, 2017.
[3] C. A. Martínez and G. I. Echeverri, "Malware Detection based on Cloud Computing integrating Intrusion Ontology representation."
[4] M. U. AbdelHameed, M. A. Sobh, and A. M. B. Eldin, "Portable executable automatic protection using dynamic infection and code redirection," in *2009 International Conference on Computer Engineering Systems*, 2009, pp. 501–507.
[5] A. Jadhav, D. Vidyarthi, and H. M., "Evolution of Evasive Malwares : A Survey," *Int. Conf. Comput. Tech. Inf. Commun. Technol. Evol.*, 2016.
[6] A. Agrawal and K. Wahie, "Analyzing and Optimizing cloud-based antivirus paradigm," *Int. Conf. Comput. Tech. Inf. Commun. Technol. Evol.*, no. Iciccs, pp. 203–207, 2016.
[7] D. Ai, G. Zeng, Y. Yue, and B. Shen, "Research of SoftMan Migration Based on Linux Checkpoint," in *2009 Fifth International Conference on Natural Computation*, 2009, vol. 3, pp. 97–100.
[8] S. Torri, W. Britt, and J. A. Hamilton, "A compiler classification framework for use in reverse engineering," in *2009 IEEE Symposium on Computational Intelligence in Cyber Security*, 2009, pp. 159–166.
[9] L. Lu, L. Qiuju, and X. Tingrong, "Research and Implementation of Compression Shell Unpacking Technology for PE File," in *2009 International Forum on Information Technology and Applications*, 2009, vol. 1, pp. 438–442.
[10] E. H. Hwang, S. J. Cho, K. J. Kim, Y. J. Kim, S. H. Yoon, and J. W. Jeon, "A recovery algorithm for PE files in a multi-core system," in *2012 12th International Conference on Control, Automation and Systems*, 2012, pp. 289–293.
[11] R. Mosli, R. Li, B. Yuan, and Y. Pan, "Automated malware detection using artifacts in forensic memory images," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, 2016, pp. 1–6.
[12] A. K. Marnerides, M. R. Watson, N. Shirazi, A. Mauthe, and D. Hutchison, "Malware Analysis in Cloud Computing : Network and System Characteristics," *Globecom 2013 Work. - Cloud Comput. Syst. Networks, Appl. Malware*, pp. 482–487, 2013.
[13] Linux Programmer's Manual, "elf - format of Executable and Linking Format (ELF) files." [Online]. Available: http://man7.org/linux/man-pages/man5/elf.5.html. [Accessed: 20-Jan-2018].
[14] Microsoft MSDN, "PE Format." [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx. [Accessed: 20-Jan-2018].