Advanced Science and Technology Letters
Vol.109 (Security, Reliability and Safety 2015), pp.14-17
http://dx.doi.org/10.14257/astl.2015.109.04

Toward an Efficient PE-Malware Detection Tool

June Ho Yang, Yeonseung Ryu

Dept. of Security Administration Engineering, Myongji University
116 Myongji-Ro, Cheoin-Gu, Yongin, Gyeonggi-Do, 449-728, KOREA
mjuboan2005@daum.net, ysryu@mju.ac.kr

Abstract. The Portable Executable (PE) format is a standard file format for executables and object code used in MS Windows operating systems. Since a number of various malwares have rapidly increased by exploiting vulnerabilities of PE structure, the need of automatic tools for PE-malware detection also increases. In this paper, we developed a PE file analysis tool using Python language for studying efficient automatic detection tool of Windows malwares.

Keywords: Portable Executable File, Malware, Static Analysis, Detection Tool

1 Introduction

Malware is a term used to describe all kinds of malicious software. There are two types of malware detection techniques according to how code is analyzed [1-6, 8-11]: First, static analysis identifies malicious code by unpacking and disassembling (or decompiling) the application [1-3]. The most important defense methods against malicious code are virus scanners. These scanners typically rely on a database of signatures that characterize known malware instances. Second, dynamic analysis identifies malicious behaviors after deploying and executing the application on an emulator or a controlled device [4-6]. The dynamic detection systems automatically load the sample to be analyzed into a virtual machine environment and execute it. While the program is running, its interaction with the operating system is recorded.

In this paper, we developed an analysis tool of Portable Executable (PE) file format as a first step to automated malware detection tool. In section 2, we describe the structure of PE file format briefly. Section 3 describes the details of our analysis tool and section 4 describes conclusion and future work.

2 Portable Executable file format

The Portable Executable (PE) format is a file format for executables, object code, DLLs, FON Font file and others used in 32-bit and 64-bit versions of Windows operating systems [7]. The PE format is a data structure that encapsulates the information

[1] This work was supported by 2015 Myongji University Research Fund.

necessary for the Windows OS loader to manage the wrapped executable code. PE is a modified version of the Unix COFF (Common Object File Format). PE/COFF is an alternative term in Windows development. Microsoft migrated to the PE format with the introduction of the Windows NT 3.1 operating system. All later versions of Windows, including Windows 95/98/ME, support the file structure.

2.1 PE File Structure

A PE file consists of a number of headers and sections that tell the dynamic linker how to map the file into memory. An executable image consists of several different regions, each of which requires different memory protection; so the start of each section must be aligned to a page boundary. For instance, typically the *.text* section (which holds program code) is mapped as execute/readonly, and the *.data* section (holding global variables) is mapped as no-execute/readwrite. However, to avoid wasting space, the different sections are not page aligned on disk. Part of the job of the dynamic linker is to map each section to memory individually and assign the correct permissions to the resulting regions, according to the instructions found in the headers.

2.2 Header

In a PE file, there are some headers. The PE header consists of a MS-DOS header, the PE signature, the file header, an optional header, and data directories. The file headers are followed immediately by section table.

The file header contains machine type, the number of sections, the time stamp, the size of optional header, and so on. The optional header is optional in the sense that some files (i.e., object file) do not have it. For image files, this header is required to provide information to the loader. The magic number determines whether an image is a PE32 or PE32+ executable. The optional header contains the size of code section, the size of initialized data section, the size of uninitialized section, the base of code, the address of the entry point, and so on. The data directory gives the address and size of a table or string that Windows uses. These data directory entries are all loaded into memory so that the system can use them at run time.

In section table, each row is a section header. The number of entries in the section table is given by the NumberOfSections field in the file header. Each section header (section table entry) has a total of 40 bytes and contains section name, the size of the section when loaded into memory, and so on.

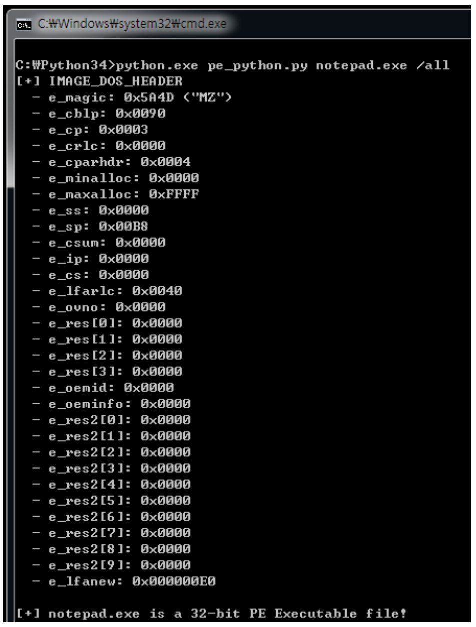
2.3 Section

Typical sections contain code or data that linkers and Microsoft Win32 loaders process without special knowledge of the section contents. A section consists of simple blocks of bytes. However, for sections that contain all zeros, the section data need not be included. The data for each section is located at the file offset that was given by the PointerToRawData field in the section header. The size of this data in the file is indicated by the SizeOfRawData field. If SizeOfRawData is less than VirtualSize, the remainder is padded with zeros. In an image file, the section data must be aligned on a boundary as specified by the FileAlignment field in the optional header.

Some sections have special meanings when found in object files or image files. Tools and loaders recognize these sections because they have special flags set in the section header, because special locations in the image optional header point to them, or because the section name itself indicates a special function of the section. For example, if a section name is ".bss", the section is an uninitialized data section.

3 Implementation Results

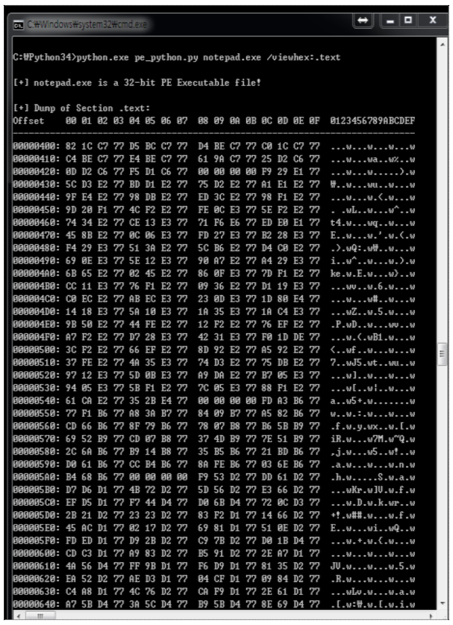
The proposed tool is composed of PE parser, malware analysis rule generator & DB, and analyzer. The PE parser decomposes a PE file into components such as headers and sections. The malware rule database has a set of rules which represent malware signatures. The analyzer compares the components of the parsed PE file with the malware rule DB to decide whether the PE file is malware or not. The example of tool usage is shown in Fig. 1. Left figure illustrates the contents of DOS header of a file named notepad.exe and right figure shows the same contents in HEX format. Our tool analyzes both the contents and the structure of PE file at the same time, and detects anomalies if they exist. In this example, there is no anomaly in the file.



```
C:₩Windows₩system32₩cmd.exe

C:₩Python34>python.exe pe_python.py notepad.exe /all
[+] IMAGE_DOS_HEADER
  - e_magic: 0x5A4D ("MZ")
  - e_cblp: 0x0090
  - e_cp: 0x0003
  - e_crlc: 0x0000
  - e_cparhdr: 0x0004
  - e_minalloc: 0x0000
  - e_maxalloc: 0xFFFF
  - e_ss: 0x0000
  - e_sp: 0x00B8
  - e_csum: 0x0000
  - e_ip: 0x0000
  - e_cs: 0x0000
  - e_lfarlc: 0x0040
  - e_ovno: 0x0000
  - e_res[0]: 0x0000
  - e_res[1]: 0x0000
  - e_res[2]: 0x0000
  - e_res[3]: 0x0000
  - e_oemid: 0x0000
  - e_oeminfo: 0x0000
  - e_res2[0]: 0x0000
  - e_res2[1]: 0x0000
  - e_res2[2]: 0x0000
  - e_res2[3]: 0x0000
  - e_res2[4]: 0x0000
  - e_res2[5]: 0x0000
  - e_res2[6]: 0x0000
  - e_res2[7]: 0x0000
  - e_res2[8]: 0x0000
  - e_res2[9]: 0x0000
  - e_lfanew: 0x000000E0

[+] notepad.exe is a 32-bit PE Executable file!
```



```
C:₩Windows₩system32₩cmd.exe

C:₩Python34>python.exe pe_python.py notepad.exe /viewhex:.text

[+] notepad.exe is a 32-bit PE Executable file!

[+] Dump of Section .text:
Offset    00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF
--------------------------------------------------------------------------------
00000400: 82 1C C7 77 D5 BC C7 77  D4 BE C7 77 C0 1C C7 77  ...w...w...w...w
00000410: C4 BE C7 77 E4 BE C7 77  61 9A C7 77 25 D2 C6 77  ...w...wa..w%..w
00000420: 0D D2 C6 77 F5 D1 C6 77  00 00 00 00 F9 29 E1 77  ...w...w.....).w
00000430: 5C D3 E2 77 BD D1 E2 77  75 D2 E2 77 A1 E1 E2 77  ₩..w...wu..w...w
00000440: 9F E4 E2 77 98 DB E2 77  ED 3C E2 77 98 F1 E2 77  ...w...w.<.w...w
00000450: 9D 20 F1 77 4C F2 E2 77  FE 0C E3 77 5E F2 E2 77  . .wL..w...w^..w
00000460: 74 34 E2 77 CE 13 E3 77  71 F6 E6 77 ED E0 E1 77  t4.w...wq..w...w
00000470: 45 8B E2 77 0C 06 E3 77  FD 27 E3 77 B2 28 E3 77  E..w...w.'.w.(.w
00000480: F4 29 E3 77 51 3A E2 77  5C B6 E2 77 D4 C0 E2 77  .).wQ:.w₩..w...w
00000490: 69 0E E3 77 5E 12 E3 77  90 A7 E2 77 A4 29 E3 77  i..w^..w...w.).w
000004A0: 6B 65 E2 77 02 45 E2 77  86 0F E3 77 7D F1 E2 77  ke.w.E.w...w}..w
000004B0: CC 11 E3 77 76 F1 E2 77  09 36 E2 77 D1 19 E3 77  ...wv..w.6.w...w
000004C0: C0 EC E2 77 AB EC E3 77  23 0D E3 77 1D 80 E4 77  ...w...w#..w...w
000004D0: 14 18 E3 77 5A 10 E3 77  1A 35 E3 77 1A C4 E3 77  ...wZ..w.5.w...w
000004E0: 9B 50 E2 77 44 FE E2 77  12 F2 E2 77 76 EF E2 77  .P.wD..w...wv..w
000004F0: A7 F2 E2 77 D7 28 E3 77  42 31 E3 77 F0 1D DE 77  ...w.(.wB1.w...w
00000500: 3C F2 E2 77 66 EF E2 77  8D 92 E2 77 A5 92 E2 77  <..wf..w...w...w
00000510: 37 FE E2 77 4A 35 E3 77  74 D3 E2 77 75 DB E2 77  7..wJ5.wt..wu..w
00000520: 97 12 E3 77 5D 0B E3 77  A9 DA E2 77 B7 05 E3 77  ...w]..w...w...w
00000530: 94 05 E3 77 5B F1 E2 77  7C 05 E3 77 88 F1 E2 77  ...w[..w|..w...w
00000540: 61 CA E2 77 35 2B E4 77  00 00 00 00 FD A3 B6 77  a..w5+.w.......w
00000550: 77 F1 B6 77 A8 3A B7 77  84 09 B7 77 A5 82 B6 77  w..w.:.w...w...w
00000560: CD 66 B6 77 8F 79 B6 77  78 07 B8 77 B6 5B B9 77  .f.w.y.wx..w.[.w
00000570: 69 52 B9 77 CD 07 B8 77  37 4D B9 77 7E 51 B9 77  iR.w...w7M.w~Q.w
00000580: 2C 6A B6 77 B9 14 B8 77  35 B5 B6 77 21 BD B6 77  ,j.w...w5..w!..w
00000590: D0 61 B6 77 CC B4 B6 77  8A FE B6 77 03 6E B6 77  .a.w...w...w.n.w
000005A0: B4 68 B6 77 00 00 00 00  F9 53 D2 77 DD 61 D2 77  .h.w.....S.w.a.w
000005B0: D7 D6 D1 77 4B 72 D2 77  5D 56 D2 77 E3 66 D2 77  ...wKr.w]V.w.f.w
000005C0: EF D5 D1 77 F7 44 D4 77  D0 6B D4 77 72 0C D3 77  ...w.D.w.k.wr..w
000005D0: 2B 21 D2 77 23 23 D2 77  83 F2 D1 77 14 66 D2 77  +!.w##.w...w.f.w
000005E0: 45 AC D1 77 02 17 D2 77  69 81 D1 77 51 0E D2 77  E..w...wi..wQ..w
000005F0: FD ED D1 77 D9 2B D2 77  C9 7B D2 77 D0 1B D4 77  ...w.+.w.{.w...w
00000600: CD C3 D1 77 A9 83 D2 77  B5 91 D2 77 2E A7 D1 77  ...w...w...w...w
00000610: 4A 56 D4 77 FF 9B D1 77  F6 D9 D1 77 81 35 D2 77  JV.w...w...w.5.w
00000620: EA 52 D2 77 AE D3 D1 77  04 CF D1 77 09 84 D2 77  .R.w...w...w...w
00000630: C4 A8 D1 77 4C 76 D2 77  CA F9 D1 77 2E 61 D1 77  ...wLv.w...w.a.w
00000640: A7 5B D4 77 3A 5C D4 77  B9 5B D4 77 8E 69 D4 77  .[.w:₩.w.[.w.i.w
```

Fig. 1. Examples of our tool usage

4 Conclusion

The PE-malwares have been varied and the number of them has been increased fast. In order to detect PE-malwares efficiently, it is required that a powerful PE analysis tool should be developed. In this work, we developed a PE analysis tool using Python. The newly developed tool can control PE file, show the contents, and detect the anomaly in it. For the next work, we will study a novel PE-malware detection technique and develop an automatic tool for managing the PE-malwares efficiently.

References

1. Moser, A., Kruegel, C., Kirda, E., "Limits of Static Analysis for Malware Detection," International Conference on Computer Security Applications, pp. 421-430 (2007)
2. Tabish, S., Shafiq, M., Farooq, M., "Malware Detection using Statistical Analysis of Byte-level File Content," ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, pp.23-31 (2009)
3. Feng, Y., Anand, S., Dillig, I., Aiken, A., "Apposcopy: Semantic-based Detection of Android Malware through Static Analysis," ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.576-587 (2014)
4. Willems, C., Holz, T., Freiling F., "Toward Automated Dynamic Malware Analysis Using CWSandbox," IEEE Security & Privacy, Vol.5, No.2, pp.32-39 (2007)
5. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T., "Graph-based Malware Detection using Dynamic Analysis," Journal in Computer Virology, Vol.7, Issue 4, pp.247-258 (2011)
6. Egele, M., Scholte, T., Kirda, E., Kruegel, C., "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools," ACM Computing Surveys, Vol.44, Issue 2, (2012)
7. Microsoft Portable Executable and Common Object File Format Specification, http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx, (2013)
8. Ye, Y., Wang, D., Li, T., Ye, D., Jiang, Q., "An Intelligent PE-malware Detection System based on Association Mining," Journal in Computer Virology, Vol.4, Issue 4, pp.323-334 (2008)
9. Choi, Y., Kim, I., Oh, J., Ryu, J., "PE File Header Analysis-based Packed PE File Detection Technique (PHAD)," International Symposium on Computer Science and Its Applications, (2008)
10. Zaidan, A., Zaidan, B., Othman, F., "New Technique of Hidden Data in PE-File with in Unused Area One," International Journal of Computer and Electrical Engineering, Vol.1, No.5, (2009)
11. Merkel, R., Hoppe, T., Kraetzer, C., Dittmann, J., "Statistical Detection of Malicious PE-Executables for Fast Offline Analysis," Lecture Notes in Computer Science, Vol.6109, pp.93-105 (2010)