Malware Detection Based on Multiple PE Headers Identification and Optimization for Specific Types of Files

Filip ZATLOUKAL, Jiri ZNOJ*

Department of Computer Science, VSB - Technical university of Ostrava,
17. listopadu 15, 708 33 Ostrava - Poruba, Czech Republic

*filip.zatloukal@vsb.cz, jiri.znoj@vsb.cz
(Received: 20-August-2017; accepted: 24-October-2017; published: 30-November-2017)
DOI: http://dx.doi.org/10.25073/jaec.201712.64

Abstract. *This paper follows our previous research where we made a basic experiment to find out if it is possible to detect malware by multiple PE header detection. The previous results show us that there is a considerable amount of malwares that connect themselves to another file. This paper summarizes our previous results, updates the results and also expands them by adding an optimization method and also by including the scan of another (specific) types of data.*

Keywords

File virus, malware detection, multiple PE headers, parasitic virus.

1. Introduction

The subject of our research is a malware detection method and testing of other related methods. The aim is to use an unconventional approach to malware detection, to test existing methods, improve them or to create a new method, if the method produces satisfactory results. The currently examined topic is malware detection by using metadata. Malware detection by metadata testing is a topic that is not often covered and this is the reason why we chose it for our research. This paper describes a detection method based on the idea of malware detection by multiple PE headers occurrence.

From our previous research, it is apparent that there are groups of malwares called parasitic viruses – they infect other files by connecting their code to the host file. They also often connect their own headers and if the malware is connected by one of the methods described below, the original file contains more than one header. We are able to detect multiple PE headers and if the scanned file goes through our optimization filter, this file is labelled as dangerous.

2. Related Work

In 2001 Schultz et al. introduced anti-malware system based on information extracted from Windows PE (Portable Executables) executables [1].

In 2006 J. Zico Kolter et al. came up with a paper with a detection and classification malicious executable. This work contains information, that thanks to detection by machine learning, when n-grams 030b0105 and 0b010219 are present, the file is malicious. After deeper investigation, they found out, that these values indicated presence of two PE headers in a single file. In their dataset, there was small portion of malicious programs with these values. Another interesting fact was, that n-gram 0000000a ap-

peared in 75 % of the malicious software. 20-25 % pieces of malware were compressed or encrypted. In their dataset, none of the benign executables were obfuscated. According to this work obfuscating could detect payload function, but it is not malware / benign detection [2].

In 2009 M. Zubair Shafiq et al. presented PE-Miner framework that extracts (189) features from PE files. Big dataset (from 2 sources) in amount of more than 15 thousand samples was used. In this framework, the detection time was decreased due to preprocessing of data against previously mentioned detections [3].

In 2010 Ronny Merkel et al. mentioned in their work 44 attributes (chosen due to other related researches) suitable for malware detection. These attributes were composed of information extracted from headers and of some overall properties (like for example string fragments). From those 44 attributes, 23 features were chosen, in one of following steps, for the classification. Feature Quality was counted for every one of those 23 features [4].

In 2011 Farrukh Shahzad et al. presented framework ELF-Miner for malware detection. Data for detection were extracted from ELF headers – Linux executables. Interesting fact about sections in this work is, that names of sections presented in benign files are ".got.plt" (49 %), ".rel.dyn" (47 %), ".comment" (3 %), ".symtab" (1 %) and the rest of known sections are not present in all tested benign files. In malware dataset collection ".got.plt" section is present only in 1 % of tested samples and ".sbss" in 6 %. This is quite different from sections in tested benign files. Other sections (with percentage occurrence) in all detected malware samples are ".rel.dyn" - 9 %, ".note" - 18 %, ".strtab" – 20 %, ".symtab" – 20 % and ".comment" – 26 % [5].

In 2012 Yibin Liao wrote a paper with PE-Header-Based detection approach with a PE-Header-Parser. This parser was used for analyzing parts of PE headers and comparing malware and benign files according to those parts of headers. The result of this detection was, that File Header has no big difference in benign files and malicious files. On the other hand, Optional Header has some interesting values. When SizeOfInitializedData equals to zero, then it is a malicious software. When DLLCharacteristics, MajorImageVersion and CheckSum are equals to zero, it is malware with 90 % accuracy. Names of malware sections could have unknown names (for example .6dnn4fh4) – benign files do have meaningful names for all sections [6].

In 2016 Mohamed Belaoued and Smaine Mazouzi focused on detect time in malware detection on PE files. They analyzed APIs (Application Programming Interfaces) and TPFs (Technical PE Features). In this work, we can find frequency of APIs and frequency of Optional-header field for malware and benign files too [7].

In 2017 David Baptiste et al. wrote a paper, where they analyzed MZ-PE binary executable files. They had the idea to focus on sections and came up with interesting results. After structural analysis of various files (malicious or not) they concluded, that only malicious files could have unnamed sections and only malicious files sometimes do not have sections at all. Another result of their analysis was, that if the name of a section had other character than alphanumeric, "." or "_", then section with such a character must be in malicious file. This work also contains rules (mentioned also in MZ-PE format documentation) for SectionAlignment, FileAlignment, SizeOfRawData, PointerToRawData VirtualSize, TimeDataStamp, PointrToSymbolTable, NumberOfSymbols, SizeOfOptionalHeader and for some other parameters. If these rules are broke, we can detect it as a malware. All legitimate files should follow these rules [8].

3. The Overview of Malware Detection Methods

1) Signature Based Detection:

It's the most commonly used detection method [9]. The method uses known patterns to detect malware. These patterns are loaded from an internal database. The method is able to detect malware fast, but it cannot detect new kinds

of threats if they are not stored in the internal database. This is the reason, why antivirus programs must upgrade their internal databases. Methods also cannot effectively deal with malware obfuscation [10].

2) Behavior Based Detection:

This method uses its own internal sandbox where the malware runs. The antivirus then analyses the actions performed by malware. This method is able to recognize both existing and new kinds of threads. A disadvantage of this method is a longer detection time needed to perform actions in the sandbox. Also, not all malware functions are performed because of application code conditional branching or not all requirements are met for malicious code activation.

3) Hybrid Analysis:

Hybrid analysis is the combination of static and dynamic analysis. In first place, static analysis is applied and then software is running in controlled environment [11].

4) Statistical Based Detection:

Statistical based method finds different attributes of software and then analyses these properties by statistic methods. Example of this method are Hidden Markov Models (HMMs) [12].

5) Different Approach:

Ashu Sharma et al. also add two more techniques: Machine Learning and Malware Normalization [13].

4. The Overview of Malware Infection Methods

One example of the malware category are parasitic viruses which use some of the following methods to copy their own code to legitimate files. Some methods can damage target file, some not - it's possible to divide the methods to two groups: destructive and non-destructive methods.

1) Prepending Viruses:

In the process of infection, the virus puts its own code in front of the original file code. If such file is executed, OS runs malicious code instead of the legitimate code first, but the original code is executed also to hide malicious behavior, so the user cannot identify it.

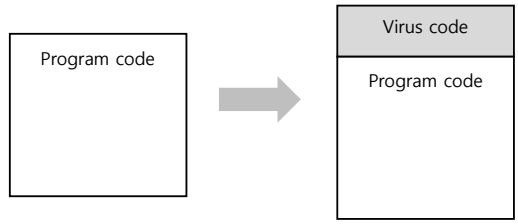

Fig. 1: The example of an infection by a prepending virus.

2) Appending Viruses:

The virus appends its code at the end of the infecting file. Because the malicious code is at the end of the file, the malware must also change the original code to assure malware activation when the file is executed.

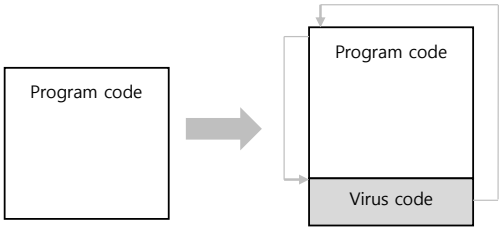

Fig. 2: The example of an infection by an appending virus.

3) Inserting Viruses:

The malicious code is placed at the address to which the entry point value points and the rest of the code is moved under the malicious code. The entry point is a value referring to the start of the executable code itself. Another way to put the code in non-destructively is to spread the code to the unused places in the file. That code doesn't have to be continuous but at the end of each section, a jump instruction must be inserted.

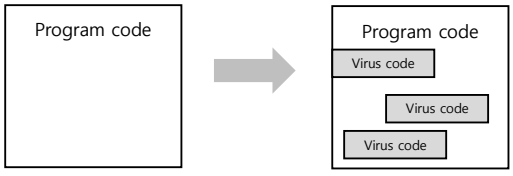

Fig. 3: The example of an infection by an inserting virus.

4) Overwriting Viruses:

Malware infects the file in a way that the original file is overwritten by the virus's own copy. Unlike others, this method is destructive. The virus can overwrite the file from the beginning but it can also choose another start location (Random Overwriting Viruses). If the executable file is overwritten from the beginning, the PE header is also overwritten and the original program will not be able to run without a header reconstruction (instead, the malware will run). If a random section of the original file is overwritten, it is possible that the program will work but it will also probably crash.

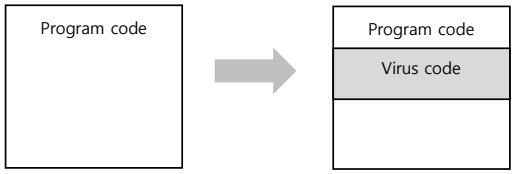

Fig. 4: The example of an infection by an overwriting virus.

5. PE Format

Portable Executable (PE) is a format defining a form and a section layout of the executable files. PE is a data structure in the binary form that is used by Windows system for exe files, dll libraries and other executable formats. PE keeps the file description information which is used by Windows OS loader and also keeps the program code itself. PE format and blocks are described below [14]:

1) DOS MZ Header:

Defined as "IMAGE_DOS_HEADER" structure. The first 64 bytes of the file is a header guarantee of DOS compatibility and it is included even in current applications. The first value of the header is named as "e_magic" (so-called magic number) and it is used to identify the DOS mode. The value must be always equal to 0x54AD ("MZ" in ASCII). The header also contains the "e_lfanew" value which is a relative offset to the PE header.

2) DOS Stub:

This is a section for the DOS code itself. Nowadays, it is compiled only to show the message: "This program cannot be run in DOS mode.".

3) PE Header:

A structure also defined as "IMAGE_NT_HEADERS" containing the signature, "IMAGE_FILE_HEADER" structure and "IMAGE_OPTIONAL_HEADER" structure. It is the main header for the Windows

executables and it consists of a variety of fields placed in "signature", "COFF header" and "PE Optional Header" structures. The description of the fields itself exceeds the scope of this paper but in the research, the "signature" value, "machine" value, "magic" value and "AddressOfEntryPoint" value are the important ones:

PE "signature" value has the similar meaning as "e_magic" value from the DOS MZ header. The value is equal to 0x50450000 ("PE \0\0" in ASCII) and it is used for the identification of PE header. The machine value holding the type of CPU the code is compiled for (the value is used for compatibility check). "Magic" field is a 2-byte value placed at the beginning of the optional header and representing the architecture type (0x010B for PE32, 0x020B for PE64, 0x0107 ROM). "AddressOfEntryPoint" is an address of the application entry point (address where the applications code begins).

4) Section Table:

Defined in "IMAGE_SECTION_HEADER" structure. Each section has its own section header and these headers are used to describe each of the following sections. The headers contain section name, relative address, section size and other values. The number of sections and their names can be different because the compiler controls these sections and names.

5) Sections:

The sections contain data created by compiler, code itself and metadata corresponding with the code. Sections names are controlled by a compiler, but the most commonly used names are [15]:

.text = section containing the main executable code; .rdata = read-only data that is globally accessible within the program; .data = global data of the program; .idata = stores the information about the import functions, if this section missing, data can be stored in the .rdata; .edata = stores the information about the export functions, if this section missing, data can be stored in .rdata section; .pdata = exception-handling for 64-bit architecture; .rsrc = stores various resources; .reloc = information for relocation of libraries.

The following picture shows the layout and the structures of PE file format:

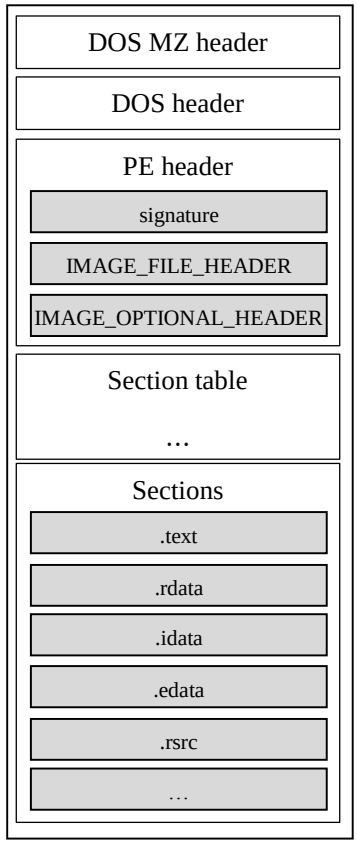

Fig. 5: Structure of PE format.

6. Experiment

The goal of our research is to scan a representative number of samples and to determine how often the malware is bound to host file by one of the described method. It will be also tested if it is possible to find malware by detecting multiple PE headers. The necessary condition is that malware connects itself (including PE header) to the code without destroying the header of the previous file. The experiment focuses on connecting the malicious code only on executable files. The goal is also to make a result opti-

mization based on the results from the previous experiment.

1) Dataset:

Our dataset is composed of 9101 Windows programs in PE file format. 5099 of them are malicious software and 4002 user-friendly (clean) executable files or DLLs. User friendly programs were collected from different sites such as Studna [16], source-forge [17], filehippo [18] and our local laboratory's files. Within research, these files were downloaded manually in order not to break the license terms of selected servers. Malicious software comes from: VirusShare [19], Malekal malware db [20] and also our private university collection.

For this research, we only process valid 32-bit or 64-bit Windows executable or DLL files with valid PE header. 16-bit DOS files or files with invalid PE headers are skipped. Most of the friendly applications are stored as compressed packages or installers. It was needed to install (or unpack) all of these applications first, because a scan of these packages/installers will return the results for the packed format instead of the unpacked application itself. Malware applications are already in the required format.

2) Multiple PE Header Detection:

For our research, a new application was developed. Application allows us to perform required actions effectively. When scanning for multiple PE headers, our software searches in the binary mode for signatures and magic values of PE headers and DOS headers. If any of the value is present, the software performs a check whether the found value is not just a binary sequence of data but if it really belongs to the PE header.

3) Malware Protection:

It is common practice to protect malware against detection and analyzing. When malware is packed, it is more difficult to use static analysis. A sample must be unpacked before. Packer takes the original malware, makes wrapper and creates a new binary file. The whole binary or only part of the file can be packed. PE header must be reconstructed for PE header analysis. [15] We connected our application to Cuckoo sandbox [21] to perform the detection of obfuscation and also data gathering from such packed applications.

7. Results

We tested package of goodware and bad samples separately. After testing all the samples, it was discovered that 504 (9.884 %) of total 5099 malware samples have multiple PE headers. It is interesting that also 231 (5.772 %) of legitimate software from the total of 4002 include multiple PE headers. After closer look on the tested malware, most of them are really connected to some host file and act as parasitic viruses. Closer look at the legitimate software shows that most of the hits (130 hits, 56.277 %) are caused by an application named "uninstall000.exe". This file is a legitimate application created by „Inno Setup" software and is used as an application uninstaller. Other hits are caused by packages including another software inside but also by a small amount of legitimate applications.

1) Optimization:

Firstly, the false positive hits from the user-friendly applications were analyzed manually. It was found out that most of those false hits were caused by the 3 types of uninstallers which cover almost all types of uninstallers and also false positive hits at all. The name of these files were always the same: unins000.exe, UNWISE.exe and Uninstall.exe. unins000.exe caused 130 (56.277 %) false hits of the total, UNWISE.exe 9 (3.896 %) false hits of the total and Uninstall.exe 4 (1.732 %).

The first step of the optimization was to exclude these types of uninstallers. We made an integrity test to check if the uninstallers with the same names are equal. We found out that even if the uninstallers have the same name and look similar, the internal data wasn't equal. It was necessary to create a mechanism that would de-

tect these types of uninstallers. It is possible to get it done by the name detection but we also added the detection by byte patterns to be sure that the uninstallers are not simply renamed for example by malware. Unique byte patterns were extracted from all mentioned uninstaller types and placed to the detection algorithm. The false positive hits were lowered from the 231 samples to 88 samples by excluding uninstallers.

Following the discovery that multiple headers appear in uninstallers mainly for user-friendly applications, it was decided to scan another type of applications: legitimate application installers. The goal of this test was to find other patterns for the optimization and to tell whether the installers also often contain PE headers.

In the test of the installers, just "exe" files (installers) were used. We made a scan of 1089 samples in total and found out that multiple PE headers are very common here. 587 samples have multiple PE headers. For optimization, the patterns were extracted by the same way as for uninstallers.

After the optimization by samples from both installers and uninstallers, malware samples were tested again to find out how many installers or uninstallers are affected by malware. The new results show that 39 samples of total 504 samples (7.738 %) are installers or uninstallers infected by a parasite virus.

These installers and uninstallers are present in both user-friendly and malware applications widely and it is difficult to distinguish them using the introduced method. Therefore, these kinds of applications were excluded from scanning.

It is a common practice to express the success of the method by ACC (Accuracy) value, but this is not appropriate for this research, because the false negative hits are not used. If a false negative appears within this method, it does not automatically mean that the hit is wrong. It can happen when the malware is just a different type of malware and not a parasitic virus. Instead, the FPR (False Positive Rate) value is used = wrongly identified goodware in percentage:

$$FPR = \frac{FP}{FP + TN} \times 100. \quad (1)$$

Where TN (True Negative) is correctly identified non-malware (goodware) and FP (False Positive) is wrongly identified goodware (as malware).

The optimization by excluding installers and uninstallers makes the FPR better: from 5.772 % to 2.199 % (lower is better). But it also decreases the number of the detected malwares from 504 to 465.

8. Discussion and Comparison

It is provable that there is a significant number of viruses connecting itself to another files, including its headers. It is possible to detect the multiple headers but this does not automatically mean that this file is a malware. After optimization, we found out, that there is 2.199 % of legitimate applications containing also multiple PE headers, by the same way as malware. Measured value can be called "false positive rate" and can be used for comparing with other methods:

Methods used by Schultz et al. [1] have following FPR values:

- RIPPER algorithm: from 5.34 % to 9.22 %.
- Naive Bayes algorithm: 3.80 %.
- Multi-Naive Bayes algorithm: 6.01 %.

Ronny Merkel et al. [4] made identification and evaluation of significant features within PE files with FPR values from: 0.9 % to 52.4 %.

M. Belaoued and S. Mazouzi [7] use "False Alarm rate" (FA) value to express false hits. With Chi-Square-Based Decision methods they received following results:

- TPFs subsets: from 4.76 % to 9.52 %
- APIs subsets: from 7.14 % to 28.57 %
- Combinations of API-TPF subsets: from 4.76 % to 9.52 %

It is obvious, that the method introduced in this paper gives satisfying results in false positive rate unlike some other methods. However, this is mainly because our method is designed for parasitic viruses only and some types of files must be excluded to get more FPR accuracy. Other described methods are used for wide range of malware.

9. Conclusion and Future Work

The experiments show that examined method can detect specific malware types that are called parasitic viruses. The problem seems to be the installers and uninstallers – these types of files are widely used by both user-friendly applications and malware. It is better to exclude these file types for the described detection method. Multiple PE detection method can be used as a base for finding malware that is in some way connected to a host file, but due to the fact that there is a small amount of legitimate applications also containing multiple headers, we cannot recommend using this method as a standalone method or as a main detection method. But we can recommend it as a fast method to label suspicious files for deeper analyzing.

There is space for improving our detecting method. We will use introduced method together with multiple other methods that would not have negative time impact. In our following research, we will try to find out, if there exist Windows API functions that are often used by malware. Scanning these functions in samples with multiple PE headers will improve results of our method. In future, malware detection will be improved by adding AI as well.

Malware writers implement mechanisms to avoid multiple infections of the same file by parasitic viruses. Following research will include a deeper look at these mechanisms to use them for malware detection together with our hereby presented method.

Acknowledgment

The following grants are acknowledged for the financial support provided for this research: Grant of SGS No. SGS 2016/175, VSB–Technical University of Ostrava and The Technology Agency of the Czech Republic TACR TF01000091.

References

[1] SCHULTZ, M., E. ESKIN, E. ZADOK and S. STOLFO. Data Mining Methods for Detection of New Malicious Executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy*. Oakland: IEEE, 2001, pp. 38–49.

[2] KOLTER, J. Z. and M. A. MALOOF. Learning to detect malicious executables in the wild. In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*. New York: ACM Press, 2004, pp. 470–478.

[3] SHAFIG, M., S. TABISH, F. MIRZA and M. FAROOG. PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In: *Recent Advances in Intrusion Detection*. Heidelberg: Springer, 2009, pp. 121–141.

[4] MERKEL, R., T. HOPPE, C. KRAETZER and J. DITTMANN. Statistical Detection of Malicious PE-Executables for Fast Offline Analysis. In: *Communications and Multimedia Security*. Heidelberg: Springer, 2010, pp. 93–105.

[5] SHAHZAD, F. and M. FAROOG. "Elfminer: Using structural knowledge and data mining methods to detect new (Linux) malicious executables. *Knowledge and information systems*. 2012, vol. 30, iss. 3, pp. 589–612.

[6] LIAO, Y. Pe-header-based malware study and detection. In: *Department of Computer Science, The University of Georgia* [online]. 2012. Available at:

http://www.cs.uga.edu/~liao/PE_Final_Report.pdf.

[7] BELAOUED, M. and S. MAZOUZI. A Chi-Square-Based Decision for Real-Time Malware Detection Using PE-File Features. *Journal of Information Processing Systems*. 2016, vol. 12, No. 4, pp. 644–660.

[8] BAPTISTE, D., E. FILIOL and K. GALLIENNE. Structural analysis of binary executable headers for malware detection optimization. *Journal of Computer Virology and Hacking Techniques*. 2017, vol. 13, iss. 2, pp. 87–93.

[9] AYCOCK, J. D. *Computer viruses and malware*. New York: Springer, 2006.

[10] BORELLO, J. M. and L. ME. Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*. 2008, vol. 4, iss. 3, pp. 211–220.

[11] UPPAL, D., V. MEHRA and V. VERMA. Basic survey on malware analysis, tools and techniques. *International Journal on Computational Sciences & Applications (IJCSA)*. 2014, vol. 4, no. 1, pp. 103–112.

[12] WONG, W. and M. STAMP. Hunting for metamorphic engines. *Journal in Computer Virology 2.3*. 2006, vol. 2, iss. 3, pp. 211-229.

[13] SHARMA, A. and S. K. SAHAY. Evolution and detection of polymorphic and metamorphic malwares: A survey. *International Journal of Computer Applications*. 2014, vol. 90, no. 2, pp. 7–11.

[14] PE Format. In: *PE Format (Windows)* [online]. 2017. Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx.

[15] SIKORSKI, M. and A. HONIG. *Practical malware analysis: the hands-on guide to dissecting malicious software*. San Francisco: No Starch Press, 2012.

[16] Studna. In: *Studna* [online]. 2017. Available at: http://www.studna.cz/.

[17] Find, Create, and Publish Open Source software for free. In: *SourceForge* [online]. 2017. Available at: http://sourceforge.net/.

[18] The Latest Versions of the Best Software. In: *FileHippo* [online]. 2017. Available at: http://filehippo.com/.

[19] VirusShare.com. In: *VirusShare* [online]. 2017. Available at: http://virusshare.com/.

[20] Malekal's forum. In: *Liste malware - malekal.Com* [online]. 2017. Available at: http://malwaredb.malekal.com/.

[21] Automated Malware Analysis - Cuckoo Sandbox. In: *Automated Malware Analysis - Cuckoo Sandbox* [online]. 2017. Available at: http://cuckoosandbox.org/.

About Authors

Filip ZATLOUKAL was born in Moravska Trebova, Czech Republic. He received his Ing. from VSB - Technical University of Ostrava in 2014. His research interests include malware analysis.

Jiri ZNOJ was born in Sumperk, Czech Republic. He received his Ing. from VSB - Technical University of Ostrava in 2016. His research interests include malware detection.