International Journal of Hybrid Information Technology
Vol.2, No.2, April, 2009

Encoded Executable File Detection Technique via Executable File Header Analysis*

Yang-seo Choi[1], Ik-kyun Kim[1], Jin-tae Oh[1], and Jae-cheol Ryou[2]
[1]*ETRI Secure Gateway System Team,* [2]*Dept. of Computer-Engineering Chung-Nam National University*
[1]*{yschoi92, ikkim21, showme}@etri.re.kr,* [2]*jcryou@home.cnu.ac.kr***

Abstract

Recently, the attack trends have been changed from fast and widespread malware propagation attacks to more sophisticated "targeted" attacks such as spy/adware, password stealers, ransom-ware, and botenets etc. and the attacks are tried via the automated malwares. In this situation, the malware is the most powerful weapon for the attackers. So, the attackers do not want their malwares to be reviled by anti-virus analyzer. In order to conceal their malware, malware programmers are getting utilize the anti reverse engineering techniques and code changing techniques such as the packing, encoding and encryption techniques. If the malware is packed or encrypted, then it is very difficult to analyze. Therefore, to prevent the harmful effects of malware and to generate signatures for malware detection, the packed and encrypted executable codes must initially be unpacked. The first step of unpacking is to detect the packed executable files. In this paper, a packed file detection technique based on a PE Header Analysis is proposed. In many cases, to pack and unpack the executable codes, PE files have unusual attributes in their PE headers. In this paper, these characteristics are utilized to detect the packed files. A Characteristic Vector (CV) that consists of eight elements is defined, and the Euclidean distance (ED) of the CV is calculated. The EDs of the packed files are calculated and represent the base threshold for the detection of packed files.

1. Introduction

As the intent of computer hackers has been changed from fast and widespread malware propagation to more sophisticated "targeted" attacks such as spy/adware, password stealers, ransom-ware, and botnets etc., malware detection has become a crucial aspect of information security [1, 2]. Actually, almost all the attacks are tried via the automated malwares which can be controlled by the attackers via the legitimated network protocols.

Currently, the most popular malware detection technique involves the utilization of signatures. Signature-based malware detection methods are very fast and effective, although they often experience scalability problems [3]. In order to detect malware using a signature, the first step involves the generation of the signature. Generating a signature involves an initial analysis of the malware, which is difficult because almost all types of malware use the anti reverse engineering techniques and the code changing techniques such as packing, encoding, encrypting, anti-revering and/or obfuscation techniques. In this paper, we are going to use the word 'encoding' and 'packing'. Actually they have the same meaning.

Based on one report, the packing ratio of malware is greater than 92% [4]. In particular, the packing of malware is the very first problem that an analyst should address. If it is impossible to unpack a packed executable file, the analysis is impossible because the codes cannot be understood.

The very first step in the unpacking of packed file is to detect packed executable files. Recently, many researchers and analysts have focused on packed file detection techniques. In this paper, however, a new lightweight packed PE file detection technique based on the characteristics of the headers of PE files is introduced.

Packed PE files were analyzed using the proposed technique. It was found that nearly every type of packed PE file with common characteristics in the PE header that differ from the normal files which are not packed can be detected. For example, with a packed file, it is necessary to unpack the packed codes to execute the intended original codes. To unpack and rewrite the codes, the code section should contain both executable and writable attributes simultaneously. Typically, however, normal PE files do not contain sections of executable and writable attributes together.

In this paper, the encoded executable file detection technique utilizes these differences between the packed and normal files. To present the different features of the packed and non-packed headers of PE files effectively, the Characteristic Vector (CV) is defined, which consists of eight elements that can show these differences effectively. It calculates the Euclidean distance (ED) [6] with the CV of a given PE file and classifies the PE file as “Packed” or “Non-Packed”. It shows very good performance, as it checks only the selected eight characteristics. Additionally, the overhead of the calculation of the Euclidean distance is very low.

This paper is constructed as follows. In the next section, related works are discussed. Section 3 summarizes the Windows PE file format. In section 4, the background is given of the author’s work on packed PE file detection. The experimental results are explained in section 5. Conclusions and a discussion of future works are given in section 6.

2. Related Works

2.1. Using entropy analysis to find encrypted and packed malware [9]

Lyda and Hamrock present an encrypted and packed malware detection technique based on entropy analysis. In their paper, they analyzed packed PE files via the byte distribution.

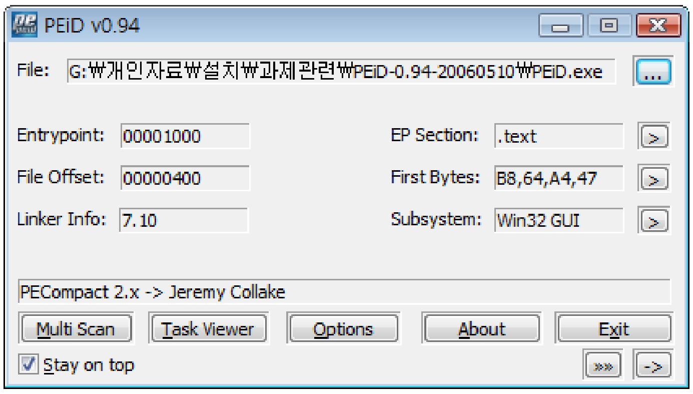
A set of metrics are developed that analysts can use to generalize the entropy attributes of packed or encrypted executable and thus distinguish them from native (non-packed or unencrypted) executables. As such, this methodology computes entropy at a naive model level, in which entropy is computed based only on the occurrence frequency of certain bytes of an executable without considering how these bytes were produced.

Entropy analysis examines the statistical variation in malware executables, enabling analysts to identify packed and encrypted samples quickly and efficiently.

2.2. PEiD[10]

PEiD is most commonly used with signature-based packers, cryptors and compilers for PE file detection. At present, it can detect more than 600 different signatures in PE files. PEiD is unique in some regard when compared to other identifiers. Its detection rates are pretty good among the current identifiers. Moreover, it has a plugin interface that supports plugins such as Generic OEP Finder and Krypto ANALyzer. Finally, it is free and easy to use.

In this paper, the packed file detection results re compared with those of PEiD, which adopts the newest packer DB.



PEiD v0.94

File: G:₩개인자료₩설치₩과제관련₩PEiD-0.94-20060510₩PEiD.exe

Entrypoint: 00001000
EP Section: .text
File Offset: 00000400
First Bytes: B8,64,A4,47
Linker Info: 7.10
Subsystem: Win32 GUI

PECompact 2.x -> Jeremy Collake

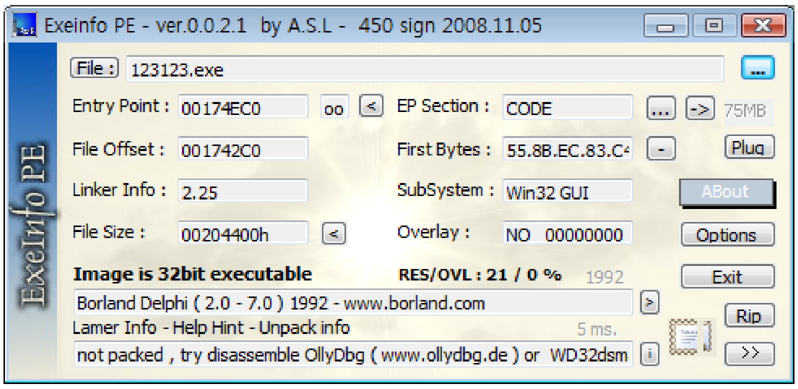
Figure 1. PEiD [10]

2.3. Ollydbg[11]

OllyDbg is a debugger that emphasizes binary code analysis, which is useful when source code is not available. It traces registers, recognizes procedures, API calls, switches, tables, constants and strings, as well as locates routines from object files and libraries. According to the program's help file, version 1.10 is the final 1.x release. Version 2.0 is in development and is being written from the ground up. The software is free of cost, but the shareware license requires users to register with the author. OllyDbg is only available in 32-bit binaries. OllyDbg shows the message box that the input file is packed when the file is detected as a packed or encrypted file.

2.4. Exeinfo PE[45]

Exeinfo PE is an ongoing work for packed PE file detection and PE header information extraction. It shows the entrypoint, file offset, compiler information and the unpack information of the input file.



Exeinfo PE - ver.0.0.2.1 by A.S.L - 450 sign 2008.11.05

File : 123123.exe

Entry Point : 00174EC0 oo
EP Section : CODE
75MB
File Offset : 001742C0
First Bytes : 55.8B.EC.83.C4
Linker Info : 2.25
SubSystem : Win32 GUI
File Size : 00204400h
Overlay : NO 00000000

Image is 32bit executable **RES/OVL : 21 / 0 %** 1992

Borland Delphi (2.0 - 7.0) 1992 - www.borland.com

Lamer Info - Help Hint - Unpack info 5 ms.

not packed , try disassemble OllyDbg (www.ollydbg.de) or WD32dsm

Figure 2. Exeinfo PE [45]

3. PE file format

There are many types of executable files. Every operating system has its own executable file formats. For example, there are PE files for Windows NT or XP, ELF files for Linux, and COM files for MS-DOS. In this paper, only the Windows PE files are addressed, as currently, most malware targets are MS-Windows systems, and the PE files represent the main route of implementation.

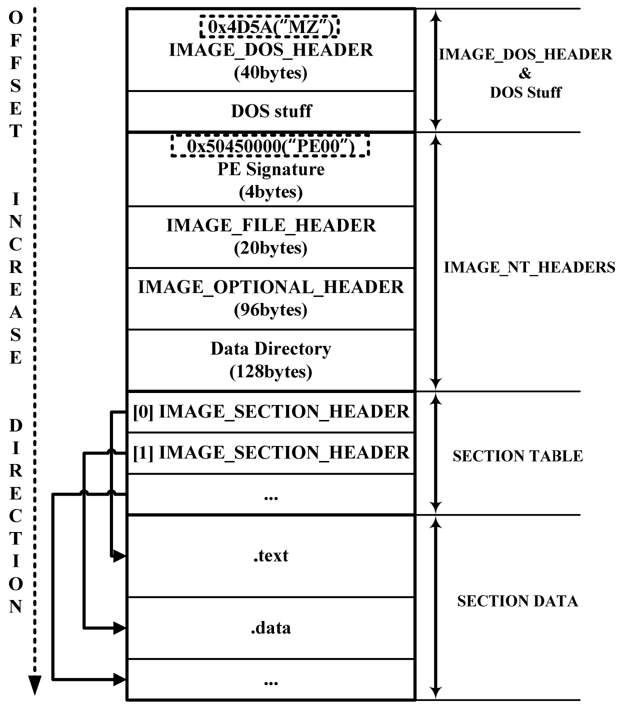


Figure 3. Windows PE File Format [7]

In Figure 3, the Windows PE file format is illustrated. As shown in Figure 3, a Windows PE file consists of four parts, a DOS header and related data, NT headers, a section table, and section data. In these parts, there are numerous attributes for PE file executions. The value of each attribute has its normal range, and some attributes are related to other attributes. Hence, one value of one attribute can be affected by the value of another attribute. In normal PE files, the relationship is very tight. For example, if the "Characteristics" attribute of a section has the "IMAGE_SCN_CNT_CODE" flag, it usually also has the "IMAGE_SCN_MEM_EXECUTE" flag, as the executable part is typically the code. However, if a file is packed, some relationships between the attributes are broken. In this paper, this feature is utilized to detect packed PE files.

4. PE Header Analysis-based packed file Detection

This paper proposes the PE file Header Analysis-based packed or encoded file Detection technique. It utilizes the differences between the attributes of normal and packed files in the PE file header.

4.1. Characteristic vectors

In order to describe the characteristics of a normal and a packed PE file, eight characteristic values (V) are selected from the attributes and status of the PE file header. With these values, the characteristic vector CV for file F is defined as follows:

$$CV_F = \{V_{F,1}, V_{F,2}, \ldots V_{F,8}\} \qquad (1)$$

Each characteristic value is depicted in Table 1.

Table 1. Characteristic Values

Characteristic Values	Description
$V_{F,1}$	The number of executable & writable sections
$V_{F,2}$	The number of the sections which is executable but not a code, or which is not executable but a code
$V_{F,3}$	The number of the section which name is not printable
$V_{F,4}$	If there is NO executable sections , it is 1
$V_{F,5}$	If the sum of every section's size is bigger than the file size, it is 1
$V_{F,6}$	If the position of the PE signature is less than the size of IMAGE_DOS_HEADER, it is 1
$V_{F,7}$	If the section of entrypoint isn't executable, it is 1
$V_{F,8}$	If the section of entrypoint isn't code, it is 1

These characteristic values are selected via heuristic analysis of the value of each field in normal and packed PE files. Upon the first analysis, the values of all the fields of the headers of normal and packed PE files are extracted and comparisons are made of the differences between the values. Some fields contain completely different values while some other values are identical. Based on the extracted values, the characteristic values are selected.

Table 2. Average Characteristic Values

	$V_{F,1}$	$V_{F,2}$	$V_{F,3}$	$V_{F,4}$	$V_{F,5}$	$V_{F,6}$	$V_{F,7}$	$V_{F,8}$
Packed Files	2.56	1.36	1.41	0.17	0.08	0.15	0.27	0.51
Non-packed Files	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The average characteristic values were fetched from 100 packed files and 100 non-packed files. The results are shown in Table 2. Table 2 shows that the differences between packed files and non-packed files are very large.

Actually each characteristic has special mean. In $V_{F,1}$, executable and writable section means that a executable section could be modified during the running time. It is the packed file's typical characteristic. Therefore if there are many executable and writable sections then we can say that the file could be the packed executable file with high possibilities. $V_{F,2}$ means that a certain section is modified intentionally for anti reversing techniques. Therefore, if $V_{F,2}$ is found then we can say the executable code is modified. $V_{F,3}$ is frequently occurred when a special pack algorithm is adopted. Basically, the section name should be a printable character string but if a special pack algorithm is adopted then the name is changed. Some pack algorithm changes the section itself, so some times it creates several sections. When a new section is created by a pack algorithm, the attributes of some sections are set as abnormal status. One of them is executable section setting. Just like $V_{F,4}$ some pack algorithm forgets to set the executable characteristic on a code section. $V_{F,5}$ and $V_{F,6}$ are about the size calculation and resizing problem of the created sections. $V_{F,7}$ and $V_{F,8}$ are about the entrypoint setting problem.

4.2. Euclidean distance

Now, we can differentiate a PE file by the Characteristic Vector CV. It means that the CV is a kind of a files signature and characteristics about how close from packed PE file. Therefore, if we can quantify the total distance of a CV, then it means that the file is close to packed PE file that much.

For the quantifying, we choose the Euclidean distance. With the CVs, the Euclidean distance (*ED(F)*) of a PE file F is defined. The definition is as follows:

$$ED(F) = \sqrt{\sum_{i=1}^{8} \left(V_{F,i}\right)^2} \tag{2}$$

The decision that a PE file is packed would be done with a threshold value. For that, the Euclidean distances of sample 100 packed PE files were calculated to determine the minimum threshold. The calculated EDs are shown in Figure 4.

The smallest ED value among those 100 packed PE files is 1.414214. Hence, the minimum threshold of the Euclidean distance of a packed file is determined heuristically as 1.4.

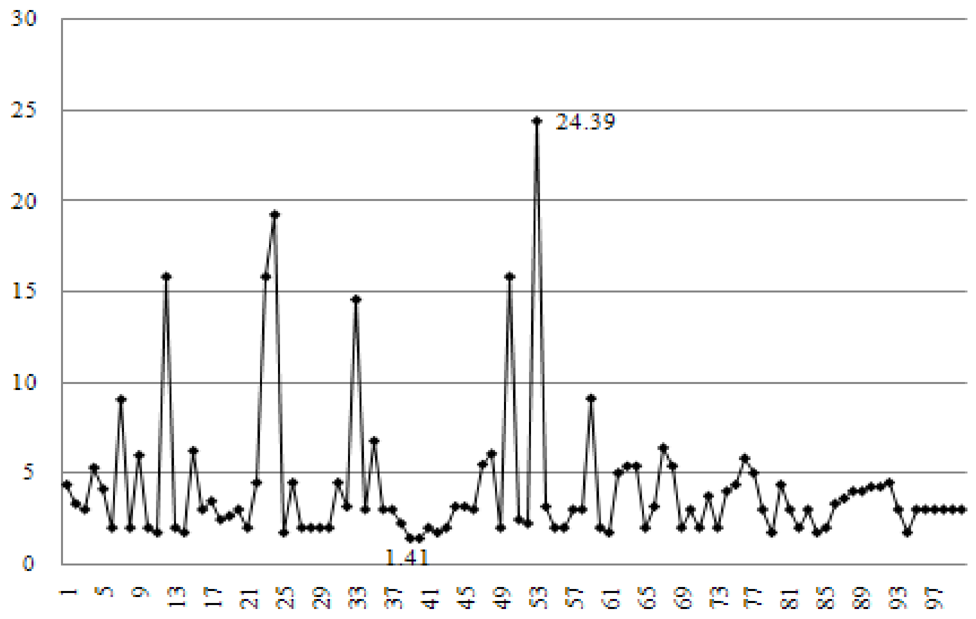


Figure 4. Euclidean Distances for 100 packed files

As we have explained, the threshold value is calculated with the sample PE files. Therefore, if the sample files are changed then the threshold could be changed. However, the selected characteristics for CV are not shown in the normal PE files. It means that even if the sample files are changed, the threshold variation would not be very big.

5. Experimental results

This approach was tested with 1,027 PE files. In those 1,027 PE files, there were 437 packed and 590 non-packed files. The test files were gathered from the “Program Files” and “system32” folders in Microsoft Windows XP operating system, downloaded from the Internet, and received from an anti-virus vendor. The PE files from the anti-virus vendor contained malware such as viruses and computer worms.

At the beginning, whether or not the files in the test sets were packed or unpacked was unknown. Therefore, all of the files in the test sets were analyzed with OllyDbg [11] and IDA Pro [12]. These tools are very helpful to inspect executable codes and determine whether a file is packed or not.

In our inspection the files are investigated manually and checked the packed files based on the section names of the PE files, some signatures about the dedicated packers on the codes, and some operation codes that are used for runtime code modification, etc. Actually the information about the packers is based on the analyzer’s personal experience. Therefore it is very difficult to formalize that is the reason why we check the test files manually.

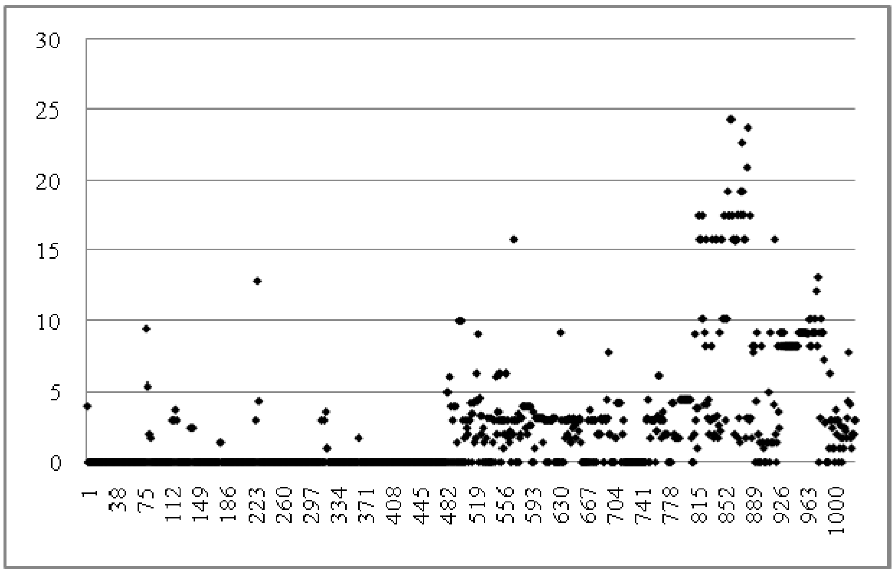


Figure 5. Euclidean Distances for the whole test set

At first, we have calculated the Euclidean distances for all the 1,027 test files. In Figure 5, all 1,027 PE files’ EDs are given.

As we mentioned before, this technique determines whether a given PE file, F, is packed or not by comparing the *ED(F)* value with the minimum threshold of packed PE files. So, we compared the calculated EDs with the minimum threshold and decided whether the files are packed or not. The results are as follows.

The total experimental results are given in Table 3. The detection rate of this technique was 93.59% and the false positive rate was 3.99%. As mentioned in section 2.2, the detection results were compared with those of PEiD.

The detection rate of PEiD was 75.06% and false negative rate was 24.94%. Although the packer database of PEiD was updated, the detection rates of this technique were higher compared to those of PEiD.

In this experiment, we assumed that the PEiD would have no false positive case, because it uses the carefully examined signatures.

Table 3. Comparison between PEiD & Our technique

	PEiD	*Our technique*
Total Test Set	1027	1027
(Packed/Not Packed)	(437/590)	(437/590)
Detection Results(Packed/Not Packed)	328/699	426/601
Detection Results(False Positives/False Negatives)	0/109	17/28
Detection Rate	75.06%	93.59%
False Positives Rate	0.00%	3.99%
False Negatives Rate	24.94%	6.41%

From this experiment, we can say that our approach has much higher packed file detection rates and much lower false negative rates. As we can see in the Table 3, the false positives rate of our technique is 3.99%. It is because all the characteristic values are selected based on the characteristics of sample files. Therefore, some characteristics could be presented in the normal files.

6. Conclusions and future works

This study presents a new PE header analysis-based packed PE file detection technique. It utilizes the fact that the PE header's features of packed files are different from those of normal non-packed PE files. For example, with packed PE files, it should be possible to modify and rewrite the executable codes. To change the codes in a PE file, the code section should contain the both executable and writable attributes. This is not the normal status of non-packed PE files.

These characteristics were selected by analyzing and comparing packed and non-packed PE files. Based on the results of the analysis eight features were selected. Using these features, the characteristic vector (CV) was defined. With the CV, it calculates the Euclidean distance of a given PE file and determines the status (packed or non-packed) of the PE file.

The packed file detection rate of this technique was approximately 93.59%, which is much higher than that of PEiD, which currently is the mostly commonly used packed file detector. The proposed technique is also very effective because it checks only the selected eight features. Additionally, the Euclidean distance calculation does not require many resources.

In the future, research on malware detection via a PE header analysis will be conducted. Essentially nearly all types of malware are packed and encrypted. Therefore, it is possible to use the proposed analysis method to detect these types of packed files.

In this paper, we utilized the Euclidean distance to quantify the amount of difference of characteristic values between normal files and packed files. However, Euclidean distance is very simple distance calculation algorithm so if we adopt other algorithms for the quantifying, then the results could be upgraded. For that, instead of Euclidean distance we are going to adopt the mahalanobis distance algorithm in the future. Because, there is a research about that if the elements of a vector have different orders then the mahalanobis distance is better than Euclidean distance to calculate the distance between the vector values – in this paper, the characteristic vectors.

Our approach checks only about the PE file's header area, so it does not need the body of the executable files. It means that it can check whether the file is packed or not only with a part of a file. This characteristic could be used for network based packed file detection technique.

We hope that this work could be helpful to analyze the malwares and can give some useful information to malware analyst.

References

[1] Bruce Schneier, "Security-FOCUS: Attack Trends", QUEUE, June 2005.

[2] Korea Internet Security Center, "Korea Internet Incident Trend Report", KISA, June 2007.

[3] Nwokedi Idike and Aditya P. Mathur, "A Survey of Malware Detection Techniques", Technical Report, Purdue University, 2007.

[4] T. Brosch and M. Morgenstern, "Runtime Packers: The Hidden Problem," Proc. Black Hat USA, Black Hat, 2006; www.blackhat.com/presentations/bh-usa-06 BH-US-06-Morgenstern.pdf.

[5] Nick Harbour, "Stealth Secrets of the Malware Ninjas", Blackhat 2007, 2007.7

[6] Wikipedia, "Euclidean Distance", http://en.wikipedia.org/wiki/Euclidean_metric

[7] Hodong Lee, "The construction and principal of executable file of Windows systems", Hanvit Media, 2005. 6

[8] C. Kreibich and J. Crowcroft. Honeycomb – creating intrustion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network, 2003.

[9] Lyda, Robert et al, "Using Entropy Analysis to Find Encrypted and Packed Malware", IEEE Security and Privacy, Apr. 2007.

[10] PEiD homepage, http://www.peid.info/

[11] OllyDbg homepage, http://www.ollydbg.de/

[12] IDA Pro homepage, http://www.hex-rays.com/idapro/

[13] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In Proceedings of the 10th Annual Computer Security Applications Conference, pages 134–144, December 1994.

[14] C. Kreibich and J. Crowcroft. Honeycomb – creating intrustion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network, 2003.

[15] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.

[16] W. Li, K.Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005.

[17] http://msdn.microsoft.com/msdnmag/issues/02/02/PE/

[18] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In Proceedings of the 7th International Symposium on (RAID), pages 201–222, September 2004.

[19] Y. M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting stealth software with strider ghostbuster. In Proceedings of the 2005 International Conference on Dependable Systems and Networks, pages 368–377, 2005.

[20] N. Weaver, V. Paxon, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In Proceedings of the 2003 ACM Workship on Rapid Malcode, pages 11–18, 2003.

[21] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. Recent Advances in Intrusion Detection (RAID), 2000.

[22] K. Ilgun, R. A. Kemmerer, and Porras P. A. State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering, 1995.

[23] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: A client-side solution for mitigating cross-site scripting attacks. In the 21st ACM Symposium on Applied Computing (SAC), 2006.

[24] A. Sulaiman, K. Ramamoorthy, S. Mukkamala, and A.H. Sung. Malware examiner using disassembled code (medic). Systems, Man and Cybernetics (SMC) Information Assurance Workshop 2005, June 2005.

[25] A. Sung, J. Xu, P. Chavez, and S. Mukkamala. Static analyzer of vicious executables (save). In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC ’04), 00:326–334, 2004.

[26] C. Taylor and J. Alves-Foss. Nate – network analysis of anomalous traffic events, a low-cost approach. New Security Paradigms Workshop, 2001.

[27] A. Vasudevan and R. Yerraballi. Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation. In Proceedings of the 29th Australasian Computer Science Conference, pages 311–320, 2006.

[28] D. Wagner and D. Dean. Intrusion detection via static analysis. IEEE Symposium on Security and Privacy, 2001.

[29] R. B. Lee, D. K. Karig, P. McGregor, and Z. Shi. Enlisting hardware architecture to thwart malicious code injection. International Conference on Security in Pervasive Computing (SPC), 2003.

[30] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.

[31] W. Li, K.Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying file types by n-gram analysis. 6th IEEE Information Assurance Workshop, June 2005.

[32] C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, and J. H. Hartman. Protecting against unexpected system calls. Usenix Security Symposium, 2005.

[33] R.W. Lo, K.N. Levitt, and R.A. Olsson. Mcf: Malicious code filter. Computers and Society, pages 541–566, 1995.

[34] W. Masri and A. Podgurski. Using dynamic information flow analysis to detect attacks against applications. In Proceedings of the 2005 Workshop on Software Engineering for secure sytems –Building Trustworthy Applications, 30, May 2005.

[35] G. McGraw and G. Morrisett. Attacking malicious code: A report to the infosec research council. IEEE Software, 17(5):33–44, 2000.

[36] M. Milenkovic, A. Milenkovic, and E. Jovanov. Using instruction block signatures to counter code injection attacks. ACM SIGARCH Computer Architecture News, 33:108–117, March 2005.

[37] A. Mori, T. Izumida, T. Sawada, and T. Inoue. A tool for analyzing and detecting malicious mobile code. In Proceedings of the 28th International Conference on Software Engineering, pages 831 – 834, 2006.

[38] J. Rabek, R. Khazan, S. Lewandowski, and R. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. In Proceedings of the 2003 ACM Workshop on Rapid Malcode, pages 76–82, 2003.

[39] San Diego Supercomputer Center. http://security.sdsc.edu/incidents/worm. 2000. 01. 18. shtml , June 2002.

[40] I. Sato, Y. Okazaki, and S. Goto. An improved intrusion detection method based on process profiling. IPSJ Journal, 43:3316 – 3326, 2002.

[41] S. E. Schechter, J. Jung, and Berger A. W. Fast detection of scanning worms infections. In Proceedings of Seventh International Symposium on Recent Advances in Intrusion Detection (RAID) 2004, 2004.

[42] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based approach for detecting anomalous program behaviors. In IEEE Symposium on Security and Privacy, 2001.

[43] R. Sekar, T. Bowen, and M. Segal. On preventing intrusions by process behavior monitoring. USENIX Intrusion Detection Workshop, 1999., 1999.

[44] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detectin network intrusions. ACM Computer and Communication Security Conference, 2002.

[45] exeinfope homepage, http://www.exeinfo.go.pl/

Authors

Yang-seo CHOI received the B.S. in Computer Science from Kang-won National University in 1996 and received the M.S. degrees in Computer Engineering from Sogang University in 2000. He is now with ETRI and in the doctorate course of Computer Engineering from Chung-nam National University. His research interests are in network security and forensics.

Ik-kyun KIM received the B.S. and M.S. degrees in Computer Engineering from Kyung-buk National University in 1994 and 1996, respectively. During 2000-2001, he stayed in PaxComm Inc. During 2004-2005, he stayed in Purdue University as a researcher. He is now with ETRI. His research interests are in information security, and network security.

Jin-tae Oh received the B.S. and M.S. degrees in Computer Engineering from Kyung-buk National University in 1990 and 1992, respectively. He is now with ETRI from 1992. From 2003, he is a team manager in ETRI. His research interests are in information security, and network security.

Jae-cheol RYOU received the M.S. in Computer Science from Iowa State University in 1988 and Ph.D. degrees in Computer Science from Northwestern University in 1990. He has been a professor at the Computer Engineering, Chung-nam National University, Korea, since 1991. He is also the chief of the Information Technology Research Center in Chung-nam National University. His research interests are in information security, network security and cryptography theory.