SCIS 2008 The 2008 Symposium on Cryptography and Information Security
Miyazaki, Japan, Jan. 22-25, 2008
The Institute of Electronics, Information and Communication Engineers

New Polymorphic Worm Detection based on Instruction Distribution and Signature

Hanyoung Noh * Jangseong Kim * Chan Yeob Yeun * Kwangjo Kim *

Abstract— The financial loss that suffered from malicious worm is known to be growing annually. In order to deal with this problem many researchers suggested the Network Intrusion Detection System (NIDS) which extracts file signature from the worms. Using the signature the NIDS checks whether received payload is worm or not. Malicious users introduced polymorphic worm that changes its signature itself to evade the NIDS, . The NIDS requires huge signatures to detect one polymorphic worm. It causes high performance overhead. Lee *et al.* [7] introduced polymorphic worm detection based on instruction distribution (PolyI-D). PolyI-D has 3-stage procedures to detect polymorphic worm. Throughout the 3-stage of PolyI-D, a payload is classified to polymorphic worm or not.

In this paper, we describe the limitation of PolyI-D and propose hybrid polymorphic worm detection. Our proposed scheme classifies a given payload into executable code and non-executable code based on instruction distribution. Only if polymorphic engine, SFX or Packer engine are found, our proposed scheme reconstructs the original file and check whether the file is worm or not using the signature. In this way our proposed scheme reduce false detection rate than PolyI-D. Moreover, our scheme has higher performance than the previous polymorphic detection schemes.

Keywords: Polymorphic Worm, Intrusion Detection System.

1 Introduction

Worms increasingly threaten users who use software that has a variety of vulnerabilities. Typical of the damage reported, caused by these computer worms, was the 50,000 employees of 13 Daimler-Chrysler assembly plants in Illinois, Indiana, Wisconsin, Ohio, Delaware and Michigan, who were idled for nearly an hour while technicians restored the computers which control the plants. To defeat the worm, Network Intrusion Detection System (NIDS) was proposed. Based on signature information such as specific code pattern of program, NIDS can verify whether the received file is worm or not. Only if the signature is same as the target program, it is classified to Internet worms. NIDS can easily defeat Internet worms effectively. However, NIDS cannot deal with worms spread speed since network professionals should extract worm signature. To solve the problem many researchers suggested automatic worm signature extracting system [1, 2, 3]. Today polymorphic worm has been introduced which can change own code pattern using its own polymorphic engine. Whenever it is distributed, polymorphic engine encrypts itself using a different key. It means that the previous IDS system needs huge signatures to detect one polymorphic worm. To detect polymorphic worm many schemes [4, 5, 6] are suggested. However most of these scheme have low performance to be used in real-time application or can be easily evaded. Based on that the polymorphic worm has a special instruction distribution on polymorphic engine, PolyI-D overcomes these problems. Throughout the 3-stage PolyI-D classifies a received payload as non-executable code and executable code. Only if the payload has executable code, PolyI-D checks whether the payload has polymorphic engine or not. However it has high false alarm rate.

In this paper we describe the limitation of PolyI-D and propose hybrid polymorphic worm detection. Our proposed scheme classifies a given payload into executable code and non-executable code based on instruction distribution. Only if polymorphic engine, SFX or Packer engine are found, our proposed scheme reconstructs the original file and check whether the file is worm or not using signature. In this way our proposed scheme reduce false detection rate than PolyI-D.

The remaining of this paper is organized as follows: In Section 2, we describe the virus, worm and polymorphic worm and introduce related work about previous IDSs and shortcoming of IDSs, and introduce high efficiency IDS, called PolyI-D overcomes shortcoming of previous schemes. In Section 3, we describe the motivation of our experiment. In Section 4, we explain our experimental work to find reasons for PolyI-D's false alarms. In Section 5, we propose our scheme 'Improved Polymorphic worm detection based on Instruction Distribution' and compare with PolyI-D. Lastly, we conclude in Section 6.

* Information and Communications University, Munji-dong, Yuseong-gu, Daejeon, 305-732 Korea, {kirseia, withkals, cyeun, kkj}@icu.ac.kr

2 Related work

In this section, we give a brief introduction about computer virus, worm, and polymorphic worm. A virus is a program code that can copy itself and infect a computer program without users permission or awareness. And, viruses can spread to other computers by infecting files on a network file system or a file system that can be accessed. However, worm is some different with virus. A worm is a self-duplicating computer program. It sends copies of itself to computer using a network and usually it does not have any user permission. Unlike a virus, it does not need to attach itself to an existing program. Worms almost cause harm to the network, if only by consuming bandwidth, while on the other hand viruses almost always corrupt or modify files on a targeted computer. Figure 1 indicates difference between virus and worms.

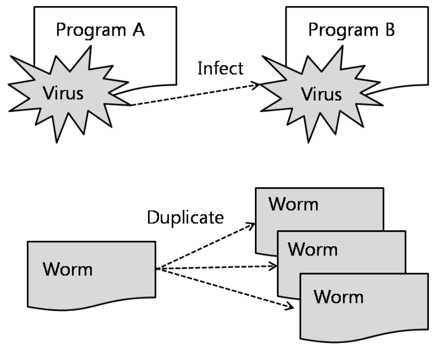


Figure 1: Operation of virus and worm

A Polymorphic worm is an advanced worm that is capable of changing program code itself to evade signature based IDS. It consists of polymorphic engine, encrypted data, and decryption key. When a polymorphic worm is executing, the polymorphic engine decrypts the encrypted data and decrypted data is executed on memory. A polymorphic worm encrypts the decrypted data itself using another key, and distributes on network when it is spread. Figure 2 shows operation of polymorphic worm.

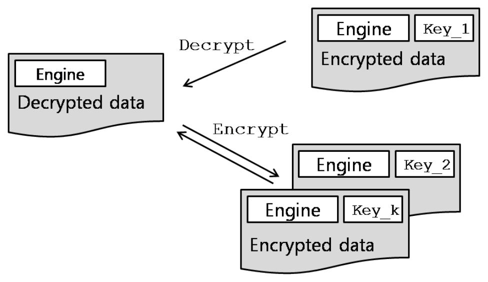


Figure 2: Operation of polymorphic worm

Since a polymorphic worm makes as many variants of itself as the length of the key size. In this reason, signature based IDS needs number of signatures as key size to detect that variant, since all variant has different signatures. It is impossible that IDS contains the whole of signatures that of polymorphic worms. That is why we need a new polymorphic detection method.

The previous IDS can detect ordinary worm with high efficiency, but it cannot response to propagation speed of worms. Many researchers suggested automatic extraction of worm signatures, for instance EarlyBird [1] and Autograph [2]. However, these schemes did not consider the polymorphic worm, so that they cannot detect polymorphic worm. Newsome *et al.* [5] proposed the Polygraph to detect polymorphic worm for the first time. The main idea of Polygraph is that the combination of invariant short contents is sufficiently unique, that it can be used as a signature. But Lee *et al.* [7] (denoted by LKHK06) analyzed that "Polygraph can be easily evaded. Only effective content is the one, corresponding to a return address of a buffer overflow attack. That is, if there were no return address in a content, the signature would not work properly, since all other contents too general. On the other hand, if there is only a return address solely as content of signature, it will work as good as a signature consisting of multiple contents. However, there can be exploits which do not need return address". Another scheme suggested by Kruegel *et al.* [4] which extracts a control flow graph, and the most frequency sub-graph is selected as a signature. They assume that the control flow of polymorphic engine is not changed too much. However this scheme has high overheads to extract the control flow signature on on-line. PolyI-D [7] is polymorphic worm detection system based on instruction distribution that overcomes weakness of the previous schemes. It has high efficiency since simple procedure that three stages classification operation. PolyI-D classifies whether the payload is polymorphic worm or not.

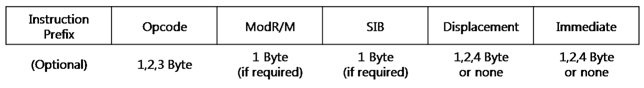
$$Score_1 = \frac{T}{D \cdot P} \tag{1}$$

T = Number of total instructions
D = Number of type of instructions
P = Number of prefixes

$$Score_2 = \frac{Score_1}{V} \tag{2}$$

V = Number of 'call' instructions

Figure 3 shows the x86 intel architecture instruction structure. We use 'Opcode' and 'Instruction prefix' for Eq.(1) and Eq.(2) of PolyI-D.



Instruction Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
(Optional)	1,2,3 Byte	1 Byte (if required)	1 Byte (if required)	1,2,4 Byte or none	1,2,4 Byte or none

Figure 3: Operation of polymorphic worm

The details of PolyI-D procedures are as follows; Firstly, PolyI-D classifies the payload as executable and non-executable code in stage 1 by Eq.(1). The payload goes to stage 2 when $Score_1$ is greater than τ_1. Secondly, PolyI-D checks the payload that contained polymorphic engine or not by Eq.(2). If $Score_2$ is greater than τ_2, then payload goes to stage 3. Finally, if payload does not contain the narrow null-bytes, that payload is classified polymorphic worm.

Note that non-executable code usually has a greater number of type of instructions and prefixes than executable code. Also a polymorphic engine uses smaller number of 'call' instructions than ordinary executable code. Therefore we can classify whether the payload has a polymorphic worm or not. The τ_1 and τ_2 are threshold values to classify the payload as executable code, non-executable code and polymorphic engine which can be determined by empirical. The following is a brief description of PolyI-D's operation.

1. If $Score_1 > \tau_1$, then go 2
2. If $Score_2 > \tau_2$, then go 3
3. If Null-bytes do not exist in payload, then it is classified as the polymorphic worm

PolyI-D has high efficiency because of its simple procedures. However it has high false alarm rate. PolyI-D mentioned reasons for false alarm rate, regular code pattern exists in non-executable code in the payload, $Score_1$ is high and $Score_2$ is high, too. If a non-executable payload has ordinary code pattern, the number of type of instructions is low and the number of 'call' instructions is low, too. Therefore this payload has higher $Score_2$ than τ_2. Namely, this payload is classified as polymorphic worm. Nevertheless, another possibility of false alarm rate may happen because the PolyI-D only detects the polymorphic engine. In other words, PolyI-D may classify a payload to polymorphic worm if the payload has a similar instruction distribution with polymorphic engine.

3 Motivation

Although PolyI-D can detect polymorphic worm efficiently, there is a false classification situations. LKHK06 pointed out that is "false positive alarms occur because some packets have specific types of data repeated". But, we expect another false alarm case because PolyI-D detect only polymorphic engine. If a program code has same structure as polymorphic worm, that program will be classified to polymorphic worm. We found some files such as Self-Extracting Archive (SFX) [10, 12], executable packer (Packer) [8, 11] which have almost similar structure of polymorphic engine. The SFX file consists of a decompress engine and a compressed data. When we execute the SFX file, it decompresses itself with decompress engine and compressed data. The Packer is almost same as SFX file, but it contains a decryption engine and an encrypted data. When it is executed, it decrypts an encrypted data in memory itself. These files should not be classified to the polymorphic worm using PolyI-D. Figure 4 shows the structure of polymorphic worm, SFX file, and Packer file.

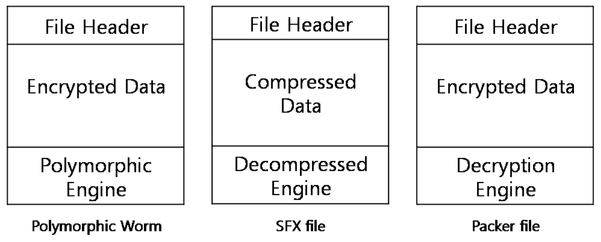


Figure 4: File structure

4 Experiment

4.1 Experiment objective

We perform an experiment to find reasons and situations of false alarms. For our experiment, we prepare non-executable codes (document, compressed file, etc), executable codes (EXE, DLL, SFX, Packer), and polymorphic worm codes. A reason for having SFX file and Packer file is they almost have same file structure as polymorphic worm. We divide our prepared codes to same size as the payload. The goal of experiment 1 is to find a situation and reason of false alarms when PolyI-D classifies the payload as non-executable code and executable code. The goal of experiment 2 is to decide whether PolyI-D can classify the SFX and Packer as polymorphic worm or not.

Our experiment environment is Windows XP SP2, and open source disassemble engine is DISIT [9]. For non-executable code disassemble, DISIT is modified a little. For SFX file, we gather executable file of installer form, ZIP, RAR, and Packer are made using UPX, Aspack. In addition, a polymorphic worm is obtained from the Internet.

4.2 Experiment result and analysis

Our experiment result 1 - Figure 6 shows the result of experiment 1. There are two cases of false alarm. First case, when non-executable code is classified as executable code, that situation is pointed out in LKHK06 which is a regular code pattern existing in non-executable code payload. However, executable code sometimes classified as non-executable code. We analyze the second case. When executable code and non-executable code are mixed in a payload, sometimes that payload is classified as non-executable code. Figure 5 shows this case.

When $Score_1$ of Case2 is greater than τ_1, this payload will have a $Score_2$ greater than τ_1, since it has a low number of 'call' instructions. It means PolyI-D will classify the payload as polymorphic worm.

Our experiment result 2 - The SFX file and Packer file can be classified to the polymorphic worm in Figure 7. In this result, we cannot set the τ_2 since SFX, Packer

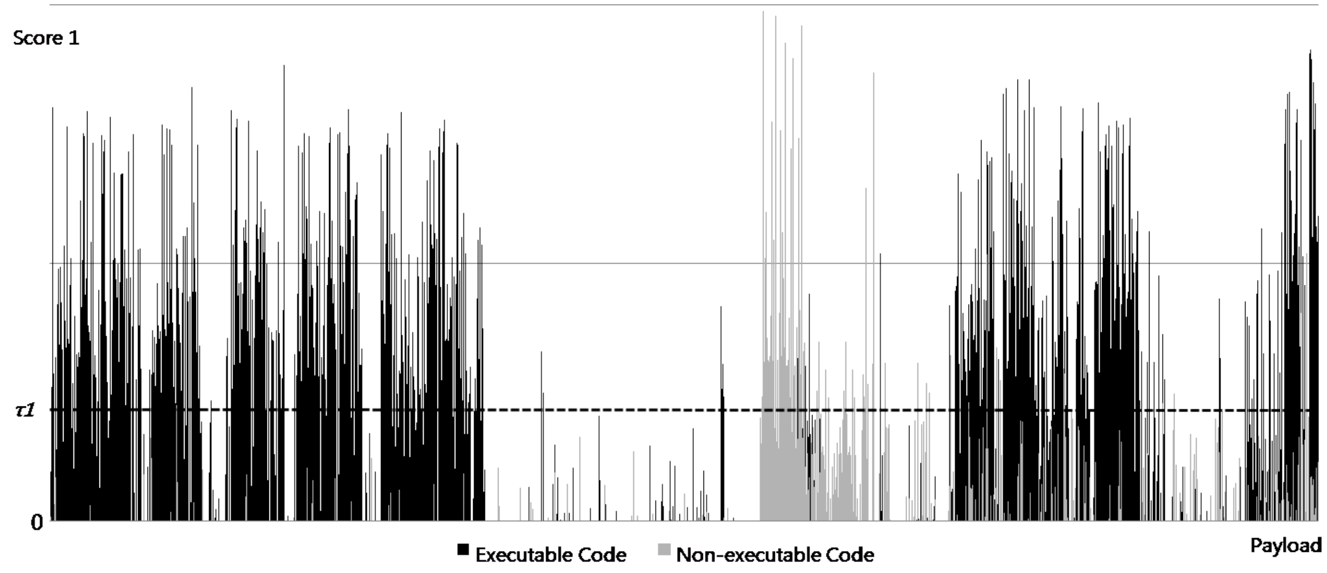


Figure 6: Result of classification executable code and non-executable code

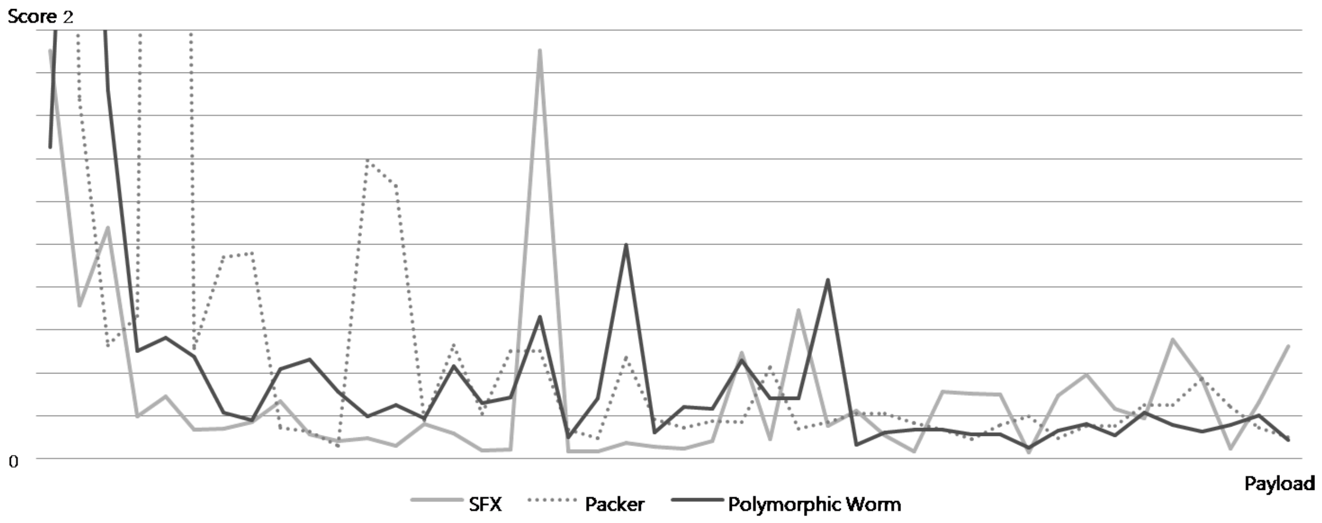


Figure 7: Result of classification executable code, SFX, Packer and polymorphic worm

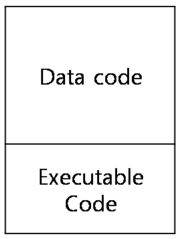


Figure 5: Mixed data code and executable code

and polymorphic worm all have high $Score_2$. This result can cause a big problem since SFX file is usually used for install program or compression program, and Packer is usually used for anti-cracking or for reducing file size. In addition, a polymorphic worm is classified as 'it is not a polymorphic worm'. When a payload is mixed with null-bytes, stage 3 of PolyI-D will classify this payload as 'it is not a polymorphic worm'. Case 1 shows mixed with null-bytes and polymorphic engine, and Case 2 shows mixed with null-bytes and divided polymorphic engine in Figure 8. These two cases are 'false negatives' in PolyI-D.

5 Our proposed scheme

To overcome the experiment result 1, which is false classification between executable and non-executable code, we classify the payload using both signature and instruction distribution. If we do not use instruction distribution, efficiency will be low. Moreover, to overcome the experiment result 2 that is false classification

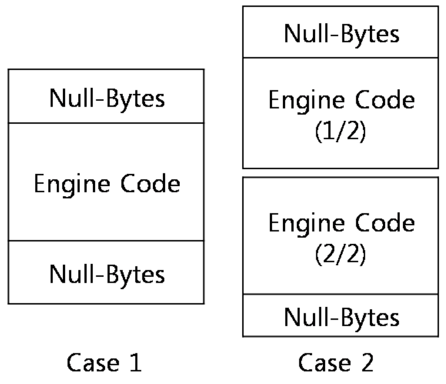


Figure 8: Mixed polymorphic engine and null bytes

with null-bytes mixed polymorphic worm payload, we remove the null-bytes in the payload before calculating $Score_1$ and $Score_2$. Our proposed scheme has 6-stages. The followings are the details of our proposed scheme.

- **Stage 1 : Null-bytes removed**

  This stage removes null-bytes in payload to overcome the null-bytes problem of PolyI-D. We already mentioned about null-bytes problem of PolyI-D in our experiment result 1, therefore we remove null-bytes first.

- **Stage 2 : Finding executable code**

  This stage is same as stage 1 of PolyI-D, but our payload does not have null-bytes. So we can classify the executable code more clearly. If we find executable code in payload at this stage, that payload goes to stage 3. However, if we cannot find an executable code, that payload is non-executable code.

- **Stage 3 : Finding polymorphic engine**

  We classify if the payload contains a polymorphic engine or not by using Eq.(2) of PolyI-D. The payload may contain a part of polymorphic engine, therefore we must check the size of payload. If payload has smaller size than ordinary payload, we can combine previous payload to new payload. A payload goes to stage 4 regardless of finding polymorphic engine.

- **Stage 4 : File construction**

  This stage works on a file, not a payload. A payload that passed the stages 1,2 and 3 is highly suspicious payload; it means that payload may part of an ordinary worm or polymorphic worm. This stage construct a file from group of payload including suspicious payload.

- **Stage 5 : File recovery**

  When stage 3 finds polymorphic engine, the file from stage 4 recovers using polymorphic engine. The word "recovery" means that decrypting the encrypted code by polymorphic engine. After stage 5, we can get the decrypted suspicious file. If cannot recover file since that file is not executable file, we can decide that file is not ordinary worm or polymorphic worm. It means 'this file is not harmful'. However, our assumption of this stage is that can decrypt the file using decrypt engine of polymorphic worm or SFX or Packer.

- **Stage 6 : Signature detection**

  This is the final stage. After stages 4 and 5, we get the executable file or decrypted file, but not the payload, therefore we apply the signature based IDS to the file. If the file accords with signature, it means 'this file is harmful'. However, if we do not find any signature according with this file, it means 'this file is not harmful' or 'unknown malware'.

Our proposed scheme uses both signature and instruction distribution for the payload and file. Firstly, it finds an executable code and a polymorphic engine using instruction distribution. Secondly, it constructs the file from payload and recovered the file using polymorphic engine when it is polymorphic worm. Finally, it detects ordinary worm or polymorphic worm by checking signature on recovered file. Figure 9 is briefly describes our proposed scheme.

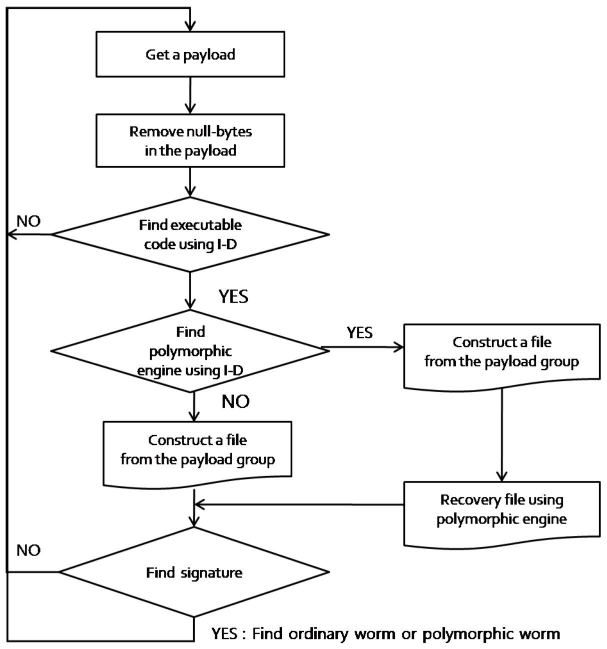

Get a payload
Remove null-bytes in the payload
NO
Find executable code using I-D
YES
Find polymorphic engine using I-D
YES
Construct a file from the payload group
NO
Construct a file from the payload group
Recovery file using polymorphic engine
NO
Find signature
YES : Find ordinary worm or polymorphic worm

Figure 9: Operation of our proposed scheme

PolyI-D can detect only polymorphic worm, however our proposed scheme can detect both ordinary worm and polymorphic worm since our scheme is based on both instruction distribution and signature. Secondly, PolyI-D has higher false alarm than our scheme because we also use signature, and not just instruction distribution of polymorphic engine. Lastly, our scheme is slower than PolyI-D as file recovery stage. Although an efficiency is slower than PolyI-D, our scheme is more faster than the previous polymorphic worm detection systems. Table 1 shows the comparison of proposed scheme with PolyI-D.

Table 1: Comparison of proposed scheme with PolyI-D

	Proposed scheme	PolyI-D
Ordinary worm detection	O	X
Polymorphic worm detection	O	O
Detection accuracy	Good	Not good
Efficiency	Reasonable	Good

O - Can detect, X - Cannot detect

6 Conclusion and Future work

A polymorphic worm is new type of worm that designed to disrupt the signature based IDS. Many researchers suggested polymorphic worm detection systems, but there is no efficient IDS. To overcome low efficiency of the previous schemes, LKHK06 proposed new IDS based on instruction distribution, called PolyI-D. Although it can detect polymorphic worm with high performance and low overhead, it has a high false alarm rate. Because PolyI-D detects just polymorphic engine since it checks only instruction distribution of payload. In this paper, we did an experiment to find the reason of false alarm rate and proposed polymorphic worm detection based on instruction distribution and signature. Our scheme checks both payload instruction distribution and file signature. In this way, our proposed scheme has high efficiency and low false alarm rate.

Our future work is to implement our scheme and to compare performance with the previous schemes. However, file recovery stage using unknown engine such as polymorphic, SFX and Packer engine is a difficult problem. We will study about decryption with unknown engine first.

References

[1] Singh, S., Estan, C., Vargohese, G., Savage,S., "The EarlyBird System for Real-time Detection of Unknown Worms," *Tech. Rep. CS2003-0761, UCSD*, Aug. 2003.

[2] H. Kim and B. Karp, "Autograph : Toward automated, distributed worm signature detection," *In 13 USENIX Security Symposium*, San Diego, California, August 2004.

[3] Singh, S., Estan, C., Vargohese, G., Savage,S., "Automated worm fingerprinting," *Proceeding of 6th symposium on Operating System Design and Implementation(OSDI)*, 2004.

[4] Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G., "Polymorphic Worm detection using structural information of executables," *8th International Symposium on Recent Advances in Intrusion Detection(RAID)*, 2005.

[5] Newsome, J., Karp, B., Song, D., "Polygraph : automatically generating signatures for polymorphic worms," *2005 IEEE Symposium on Security and Privacy*, 2005.

[6] Jisheng Wang, Ihab Hamadeh, George Kesidis, and David J.Miller, "Polymorphic Worm Detection and defence : System Design, Experimental Methodology, and Data Resources," *SIGCOMM'06 Workshops*, Pisa, Italy, September 11-15, 2006.

[7] Ki Hun Lee, Yuna Kim, Sung Je Hong, Jong Kim, "PolyI-D : Polymorphic Worm Detection Based on Instruction Distribution," *WISA 2006*, Jeju Island, Korea, August 28-30, 2006.

[8] Aspack, http://www.aspack.com/

[9] DISIT, Open Source Disassembler Engine, http://www.piotrbania.com/all/disit/

[10] RARLAB, http://www.rarlab.com/

[11] UPX, the Ultimate Packer for eXecutables, http://upx.sourceforge.net/

[12] WinZip, http://www.winzip.com/