INTERNATIONAL JOURNAL OF COMPUTERS
Issue 1, Volume 3, 2009

Validation Methods of Suspicious Network Flows for Unknown Attack Detection

Ikkyun Kim, Daewon Kim, Yangseo Choi, Koohong Kang, Jintae Oh and Jongsoo Jang

***Abstract*— The false rate of the detection methods which are based on abnormal traffic behavior is a little high and the accuracy of the signature generation is relatively low. Moreover, it is not suitable to detect exploits and generate its signature. In this paper, we have presented ZASMIN (Zeroday-Attack Signature Management Infrastructure) system, which is developed for novel network attack detection. This system provides early warning at the moment the attacks start to spread on the network and to block the spread of the cyber attacks by automatically generating a signature that could be used by the network security appliance such as IPS. This system have adopted various technologies — suspicious traffic monitoring, attack validation, polymorphic worm recognition, signature generation — for unknown network attack detection. Especially, the validation functions in ZASMIN have to able to cover 1) polymorphism, which is an encrypted attack code at the penetration and operation step, 2) executables, which are any binary functions at each step, and 3) malicious string. And also, we introduce two concepts to validate the pre-processing of the suspicious traffic. The one is attack-based validation and the other is signature-based validation. These validation functions can reduce the false rate of the unknown attack detection. In order to check the feasibility of the validation functions in ZASMIN, we have installed it on real honeynet environment, then we have analyzed the result about detection of unknown attack. Even though short–period analysis is not enough long to detect various unknown attacks, we confirmed that ZASMIN can detect some attacks without any well-known signature.**

***Index Terms*— Zero-day Attack, Signature, cyber attack, Intrusion Detection**

I. Introduction

Protecting network systems against recurrent and new worm attacks is a pressing problem. Current solutions focus on end system patching, the purview of end user vigilance with system support for streamlining the patching effort, and well-trained intrusion prevention system at network perimeter. For both technical (e.g., disruption, unreliability, irreversibility) and non-technical (e.g., awareness, laxity) reasons [34], [37], software and signature patches incur a significant time lag before they are adopted, and even then, the deployment level is partial. According to recent studies [30], [29], an average of twenty to forty new vulnerabilities in commonly used networking and computer products are discovered every month. Such wide-spread vulnerabilities in software add to today's insecure computing/networking environment. Similar new vulnerabilities in networks and applications are discovered and published on a daily basis. This insecure environment has given rise to the ever evolving field of intrusion detection and prevention.

In the classification point of view of the typical network intrusion detection methodology, we can consider the "zero-day" worm problem as the extension of the anomaly detection methodology. However, as the zero-day network attack became more sophisticated and faster in spreading across network, it differs from the existing anomaly detection methodology and researches. In the initial of this research of a field, it was initiated from the detection and signature generation methods [26], [31], [32] using the content prevalence model which considers the propagation of the super worm including Code-Red, Slammer, etc. But the false rate of the detection methods which are based on it is a little high and the accuracy of the signature generation is relatively low. Moreover, it is not suitable to detect exploits and generate the signature, if we look into the recent trend of new network attacks. For example, after Sasser worm occurred in 2004, the network attack of the similar type markedly decreases. And malicious software mainly spread by using E-mail, downloader, dropper, and etc. Therefore, as to researches [26], [16] using the property of the similarity or the repeatability of the network traffic, the effectiveness decreases, while some static or dynamic analysis method of network packet have gotten the attention in detecting the malicious software.

In this paper, we have developed the Zeroday-Attack Signature Management Infrastructure(ZASMIN) system for novel network attack detection. This system provides early detection function and validation of attack at the moment the attacks start to spread on the network. The system could also contain the spread of the cyber attacks by automatically releasing a signature that could be used by the network security appliance such as IPS. In order to detect unknown network attack, the ZASMIN system has adopted various of new technologies, which are composed of suspicious traffic monitoring, attack validation, polymorphic worm recognition, signature generation. Some of these functionalities are implemented with hardware-based accelerator to be able to deal with giga-bit speed traffic, therefore, it can be applicable to Internet backbone or the bottle-neck point of high-speed enterprize network without any loss of traffic. After we installed the ZASMIN on real honey-net environment in the internet exchange point (IX), we have analyzed the results of the ZASMIN about detection of unknown attack for two days. Even if two-day analysis

Manuscript received December 28, 2008; This work was sponsored by the Korea Ministry of Knowledge and Economy under the Zero-day Attack Signature Management INfrastructure (ZASMIN) Project [2006-S-042-03].

Ikkyun Kim, Daewon Kim, Yangseo Choi, Jintae Oh and Jongsoo Jang are with Information Security Research Division, ETRI, Gajeong-dong, Yuseong-gu, Daejeon, 305-700, South Korea. (Email: {ikkim21, dwkim77, yschoi92, showme, jsjang}@etri.re.kr)

Koohong Kang is with Department of Information and Communications Engineering, Seowon University, South Korea. (Email: khkang@seowon.ac.kr)

is not enough long to detect various unknown attacks, any experimental report on the zero-day attack detection in the real network environment have never presented, and also we could find some attacks without any well-known signature and infer the tendency of the network attack at this point in time through the case study.

The rest of the paper is organized as follows. In section 2, we talk about the zero-day attacks and in the next section, we deal with the related works. In section 4, we introduce the ZASMIN system and describe the detail methodologies adopted. In section 5, we shortly explain the real test environment including honeynet, and we analyze the two-days result of the ZASMIN system. Finally, we conclude the paper and discuss the future work.

II. Zero-day Attack Detection

As it is mentioned in the vulnerability's law [29], new vulnerabilities are released with every year and the infection rate of new network attack is faster than our prediction. That is, whilst it activates the automatic patch function, the threats of the zero-day network attack are increased as times go by. As the gap between the release time of the vulnerability and the outbreak time of the attack using the vulnerability gradually decreases, the necessary time to response its vulnerability reached zero-day and zero-time. Actually in case of Slammer worm in 2003, after the server buffer overflow vulnerability was released, its exploit occurred in 185 days (called 185-day attack). And in case of Sasser worm in 2004, it took 19 days (called 19-day attack). At last, the attack (zero-day attack) using the vulnerability which is not released was occurred in the Mozilla Firefox application in 2005. Furthermore, recently the research on the detection of zero-day network attack and the signature generation is highlighted as an issue. In the classification point of view of the typical network intrusion detection methodology, we can consider the "zero-day" worm problem as the extension of the anomaly detection methodology. However, as the zero-day network attack became more sophisticated and faster in spreading across network, it differs from the existing anomaly detection methodology and researches. In the initial of this research of a field, it was initiated from the detection and signature generation methods [26], [31], [32] using the content prevalence model which considers the propagation of the super worm including Code-Red, Slammer, etc. But the false rate of the detection methods which are based on it is a little high and the accuracy of the signature generation is relatively low. Moreover, it is not suitable to detect exploits and generate the signature, if we look into the recent trend of new network attacks. For example, since Sasser worm occurred in 2004, the network attack of the similar type markedly has decreased. And malicious software mainly spread by using E-mail, downloader, dropper, and etc. Therefore, as to researches [26], [16] using the property of the similarity or the repeatability of the network traffic, the effectiveness decreases, while some static or dynamic analysis method of network packet have gotten the attention in detecting the malicious software.

III. Related Works

This paper is mainly related to two fields in the anomaly attack detection category. The one is statistical methods for anomaly detection; The other is machine learning method for it; In the following, we first briefly review the first which are less close to this work. Then we will focus on comparing our work with the latter.

A. Statistical Approach

Statistical approaches to anomaly detection have some advantages. Firstly, these systems, like most anomaly detection systems, do not require prior knowledge of security flaws and/or the attacks themselves. As a result, such systems have the capability of detecting "zero day" or the very latest attacks. In addition, statistical approaches can provide accurate notification of malicious activities that typically occur over extended periods of time and are good indicators of impending denial-of-service (DoS) attacks. A very common example of such an activity is a portscan. Typically, the distribution of portscans is highly anomalous in comparison to the usual traffic distribution. However, statistical anomaly detection schemes also have drawbacks. Skilled attackers can train a statistical anomaly detection to accept abnormal behavior as normal. It can also be difficult to determine thresholds that balance the likelihood of false positives with the likelihood of false negatives. In addition, statistical methods need accurate statistical distributions, but, not all behaviors can be modeled using purely statistical methods [28]. One of the earliest intrusion detection systems was developed at the Stanford Research Institute (SRI) in the early 1980's and was called the Intrusion Detection Expert System (IDES) [8], [20]. IDES was a system that continuously monitored user behavior and detected suspicious events as they occurred. In IDES, intrusions could be flagged by detecting departures from established normal behavior patterns for individual users. As the analysis methodologies developed for IDES matured, scientists at SRI developed an improved version of IDES called the Next-Generation Intrusion Detection Expert System (NIDES) [2], [3]. Statistical Packet Anomaly Detection Engine (SPADE) [9] is a statistical anomaly detection system that is available as a plug-in for SNORT [33], and is can be used for automatic detection of stealthy port scans. SPADE was one of the first papers that proposed using the concept of an anomaly score to detect port scans, instead of using the traditional approach of looking at p attempts over q seconds. In conjunction with an automated worm detection and signature generation system such as EarlyBird [32] and Autograph [14], signatures of new worms may be uploaded onto worm filters, perhaps in time to mitigate—if not prevent—significant damage and wide-spread infection.

B. Machine Learning

In static anomaly-based detection, characteristics about the structure of packet or the program under inspection are used to detect malicious code. A key advantage of static anomaly-based detection is that its use may make it possible to

detect malware without having to allow the malware carrying program execute on the host system. We categorize the static anomaly detection into four parts : protocol header analysis, packet payload analysis, application control detection and static analysis of binaries.

1) Protocol Header Analysis : Mahoney et al. [21], [22], [23] presented several methods that address the problem of detecting anomalies in the usage of network protocols by inspecting packet headers. The common denominator of all of them is the systematic application of learning techniques to automatically obtain profiles of normal behavior for protocols at different layers. Packet Header Anomaly Detector (PHAD) [21], LEarning Rules for Anomaly Detection (LERAD) [22] and Application Layer Anomaly Detector (ALAD) [23] use time-based models in which the probability of an event depends on the time since it last occurred. Taylor and Alves-Foss [35] propose a computationally low cost approach to detecting anomalous traffic. Their approach is referred to as the Network Analysis of Anomalous Traffic Events (NATE). This technique focuses on attacks which exploit network protocol vulnerabilities. In [41], protocol type and Logistic Regression were used to pick up the feature sets which can get nearly the same performance as the full feature using a Support Vector Machine.

2) Payload Analysis: Kruegel et al. [17] show that it is possible to find the description of a system that computes a payload byte distribution and combines this information with extracted packet header features. In this approach, the resultant ASCII characters are sorted by frequency and then aggregated into six groups. Maxion and Feather [24] characterized the normal behavior in a network by using different templates that were derived by taking the standard deviations of Ethernet load and packet count at various periods in time. An observation was declared anomalous if it exceeded the upper bound of a predefined threshold. Lee and Xiang [18] used several information-theoretic measures, such as entropy and information gain, to evaluate the quality of anomaly detection methods, determine system parameters, and build models. These metrics help one to understand the fundamental properties of audit data. Wang and Stolfo [39], [38] present PAYL, a tool which calculates the expected payload for each service (port) on a system. A byte frequency distribution is created which allows for a centroid model to be developed for each of the hosts services. Li et al. [19] describe Fileprint (n-gram) analysis as a means for detecting malware. During the training phase, a model or set of models are derived that attempt to characterize the various file types on a system based on their structural (byte) composition. These models are derived from learning the file types the system intends to handle. The authors premise is that benign files have predictable regular byte compositions for their respective types. In [1], its method generates normal signature sequence and alignment threshold value from processing the system training data and encode observed network connection into corresponding DNA nucleotides sequence, then to align the signature sequence with observed sequence to find similarity degree value and decide whether the connection is attack or normal.

3) Application Control Detection: Hittel [10] showed how a metamorphic sled can be constructed and in the same paper, developed Snort rules for detection; however, their number can be very large. Toth and Kruegel [36], also concentrating on the NOOP sled, went one step further. They used binary disassembly to find sequences of executable instructions bounded by branch or invalid instructions; hence, longer the sequence, greater the evidence of a NOOP sled. However, this scheme can be easily defeated by interspersing branch instructions among normal code [10], resulting in very short sequences. Recently, Pasupulati et al. [27] proposed a technique to detect the return address component by matching against candidate buffer addresses. While this technique is very novel and perhaps the first to address metamorphic and polymorphic code, there are caveats. First, the return address component could be very small so that when translated to a signature, it is not specific enough. Secondly, even small changes in software are likely to alter buffer addresses in memory. Consequently, this approach runs into similar administrative overheads as existing signature-based detection systems.

4) Static Analysis of Binaries: Bergeron, et al. [5] propose a method that attempts to analyze the malicious intent of an executable before it is executed. This form of malware detection makes use of static analysis to identify the malicious code. And also, Bergeron et al. [6] propose a specification-based detection method. First the code is disassembled into assembly code. Then to make the assembly code easier to analyze, it is transformed into a higher level representation. Flow analysis is used to help create the high level abstraction. Suspicious APIs of the PUI are identified based on the behavioral specification of the system. This work differs from [5] primarily in its specification of how to derive a high-level imperative representation and its use of program slicing. Andersson et al. [4] proposed a search algorithm to detect the executable code transmitted in buffer overflow attacks. However, the algorithm only identified the operation of the buffer overflows attack by printing out the sequence of system calls used in the exploit. Moreover, Chinchani and Berg [7] proposed a fast static analysis approach to detect exploit code inside network flows, where they relied on the control and data flow analysis at instruction level. Unfortunately, it still not fast enough to handle very large network traffic as mentioned in the paper. The idea of SigFree [40] is motivated by an important observation that “the nature of communication to and from network services is predominantly or exclusively data and not executable code” [7].

IV. The ZASMIN System

We have developed the Zero-day Attack Signature Management Infrastructure (ZASMIN) system for novel network attack detection. This system provides early detection function and validation of attack at the moment the attacks start to spread on the network. The system could also contain the

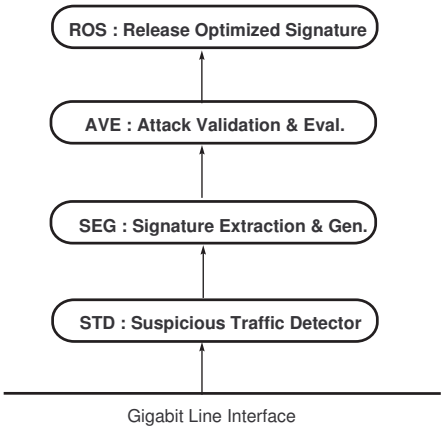


Fig. 1: The Workflow Diagram of the ZASMIN.

spread of the cyber attacks by automatically releasing a signature that could be used by the network security appliance such as IPS. As shown in the figure 1, in order to detect unknown network attack, the ZASMIN system has adopted various of new technologies, which are composed of suspicious traffic monitoring, attack validation, signature generation. Some of these functionalities are implemented with hardware-based accelerator to be able to deal with giga-bit speed traffic, therefore, it can be applicable to Internet backbone or the bottle-neck point of high-speed enterprize network without any loss of traffic. In the following, we describe the detail methodologies used in the for each blocks.

A. STD : Suspicious Traffic Detection

The functionality to classify the suspicious traffic in the normal traffic is very prerequisite in the traffic-based statistical detection approach. The STD in the ZASMIN monitors all of the traffic on the line without any loss and periodically notices the traffic information, that is, dispersion of destination IP address , TCP connection trial count, TCP connection success count and stealth scan trial count. These traffic information serves as the criteria to determine the suspicious traffic flow. Most of worms, which have features called 'fast propagation attack' or 'scan-based attack', should be filtered with these metric standard. Basically, the thresholds to determine the suspicious traffic depend on the type of network, which could be the honey-net configuration, IDC network with many web server, the transit point of the enterprize network, and so on. In this experiment of the global honeynet, we set the threshold value of the address dispersion as 5 degree — the address dispersion means that a source IP with same protocol and destination port connect to how many destination IP —, where measurement period is 8 seconds. The determination value of the TCP session success rate is non-zero. In case that TCP session success rate is zero, the $\langle$ *Source IP, Protocol, Destination Port* $\rangle$ 3 tuple-based flow doesn't contain the payload as a connection trial traffic like SYN packet. We can say that these packets are very supicious, but they don't only effect on the actual compromise of the victim besides DOS attack, but also the system can not generate or validate new signature of the attack.

B. SEG : Signature Extraction & Generation

The goal of the SEG in the ZASMIN is to generate the candidate of signature, which should be the unique pattern to be able to symbolize the attack packet. The SEG in the ZASMIN is based on prevalence message through the content analysis of payload like EarlyBird [32] and Autograph [14]. First, the SEG analyzes the flow pattern of the contents in the suspicious packet from the STD. The flow means the $\langle$ *Source IP, Protocol, Destination Port* $\rangle$ 3 tuple-based flow. And it extract the patterns which is seems to an attack. At this time, it use the same concept – prevalent content sampling– as Autograph [14]. It extracts the fixed byte content with overlapped windows, that is, it extracts 3-bytes contents at the x offset position, it again extract 3-bytes contents at the $x+1$ offset position on the next time. The method of the sampling the suspicious pattern is depend on the number of destination of packets which are contained the fixed size content. And it also use the ASCII filter, which exclude the part of payload composed of ASCII code pattern. It is based on the assumption that the payload content composed of ASCII code pattern has very low probability to be attack. Finally, it reconstructs the byte strings of flow pattern to a signature of the suspicious packet. It considers the relationship between the byte strings, so it proceed clustering byte strings with a resemblance, $R(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$, and a containment $C(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$. It consider the relationship between the byte strings, so it proceed clustering byte strings with the resemblance and the containment, then, finalize the signature candidate with LCS (Longist Common Substirng) clustering method.

C. AVE : Attack Validation & Evaluation

Basically, the suspicious traffic from the STD tend to have high false positive rate. Therefore, we have to validate the packets of the suspicious traffic. The functions of the validation depend on the feature of attack packets transmitted on the network. The feature is shaped on the specific step during the lifetime of network exploits as shown figure 2. The specific step could be the penetration or operation step in the propagation phase. At this phase, any shellcode could be delivered to the victim target, and some malicious activity is performed at operation step. The features could be monitored on the network at two steps are the penetration code and operation code. The penetration code would contain incomplete binary of exploit code or shellcode, and its type could be normal executable binary code or polymorphic with any encrypted code. The operation code is the complete binary of worm code or virus file, and its type also could be normal or polymorphic. Therefore, the validation functions in the AVE have to able to cover 1) polymorphism, which is an encrypted attack code at the penetration and operation step, 2) executables, which are any binary functions at each step, and 3) malicious string.

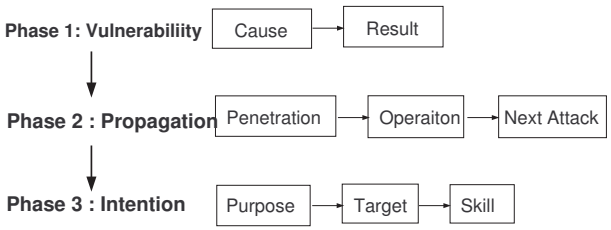


Fig. 2: The life time of the network exploit

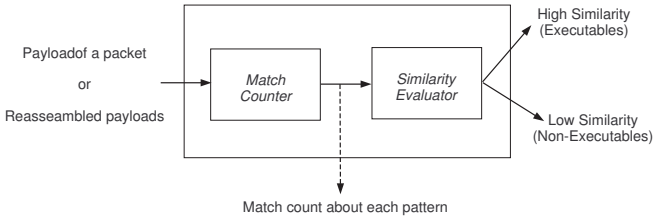


Fig. 3: The workflow of the function call-based detection.

We introduce two concepts to validate the previous processing of the STD and SEG. The one is *attack-based validation* and the other is *signature-based validation*. The attack-based validation is the thing whether the session traffic which trigger to make new attack pattern contains any malicious component or not. And The signature-based validation is the thing whether the session traffic which triggered by newly generated signature contains any malicious feature or not. We can say that the former is a validation concept about the past and current traffic, the latter is a validation one about future suspicious traffic using newly-made signature. That is, each concept deals with just different input traffic and applies same methodologies for its validation. The AVE in the ZASMIN has various methodologies to validate the suspicious traffic.

1) Recognition of Executables: In the process to detect the malicious code on the network, the key element is to determine the presence of the executable code within a payload. The method to distinguish the program-like payload from noncode data and non-exploit code, that is, in order to decide the presence of the executable code, we assume that all of payload are executable. Because Intel instruction set (IA-32) is simple structure, its codes can't be distinguished from data bytes by using only the disassembly method. If the static disassembling is performed about all byte data, the complete assembly code can be generated, and if the emulation of the execution level is not performed, it is very difficult to determine that it is executable code. Because the Intel IA32 OP-Code uses the CISC format, all OPcode encodings can be expressed as 1byte (256 encodings) size. Therefore all kinds of data, which are the ASCII format or the binary, can be translated into the assembly language with the IA-32 static disassembler. It means that it is very difficult to determine whether it is executable or not through the syntax examination of the disassembled code. The challenge of this thesis is to look for the method for determining the executable part within network payload without the emulation of the execution level.

The AVE has two parallel detectors for recognize any executable binary code in reassembled payload. The AVE has two parallel detectors , FCED (Function Call-based Executable code Detection) [12] and ICED (Instruction Coloring-based Executable code Detection) [15], which provide the functionality of high-speed recognition of executable codes by eliminating the emulation of the execution level. The presented methods can detect the executable shellcodes and executable parts in any types of files and also they would be complementary to improve the detection rate. The first method (FCED) detects the instruction patterns of function call mechanism within executable codes; that is, this function call mechanism must be an obvious feature of executable codes. For this, the FCED calculates the matching probabilities of five instruction patterns – ec, pc, pm mpr, pr and lr – which are corresponding to the function call mechanism, and then determines the existence of executable codes in packets using the similarity evaluator as shown Figure 3. We also explore the optimization of parameters including the detection window size, allowable instruction gap, minimal matching trial count and decision threshold to improve the detection rate.

TABLE I: Instruction patterns according to the function call mechanism. 'esp ops.' means instructions that include '%esp'.

Function	Notation	1	2	3
Fn. Call	ec	eps ops.	**call(s)**	
	pc	push	**call(s)**	
Fn. Start	pm	push ebp	**mov ebp, esp**	
Fn. Return	mpr	**mov esp, ebp**	pop ebp	ret(s)
	pr	**pop ebp**	ret(s)	
	lr	**leave**	ret(s)	

Whilst the FCED detects most shellcodes in malwares, it could fail to identify in the case that the attack codes do not contain the pattern of the function call mechanism. In order to overcome this drawback, we propose another method called ICED that uses an instruction spectrum analysis methodology; that is, each instruction set is represented as a unique color, and then the whole sequence of instructions is analyzed by the translated color pattern. In order to generalize it, we introduce an instruction transition probability matrix (ITPX) which is comprised of the IA-32 instruction sets and reveals the characteristics of executable code's instruction transition patterns. Finally, we use a simple algorithm to detect executable code inside network flows using a reference ITPX which is learned from the known Windows Portable Executable (PE) files.

In order to find the executable part within a payload, we propose the instruction spectrum analysis of which each instruction set is represented its color and then the whole sequence of instructions is analyzed by the translated color pattern. We assume two things; (i) there is the tendency that some instruction set are consecutively repeated and others are not so, (ii) in terms of the context between an instruction group and the other instruction group, there is any correlation pattern.

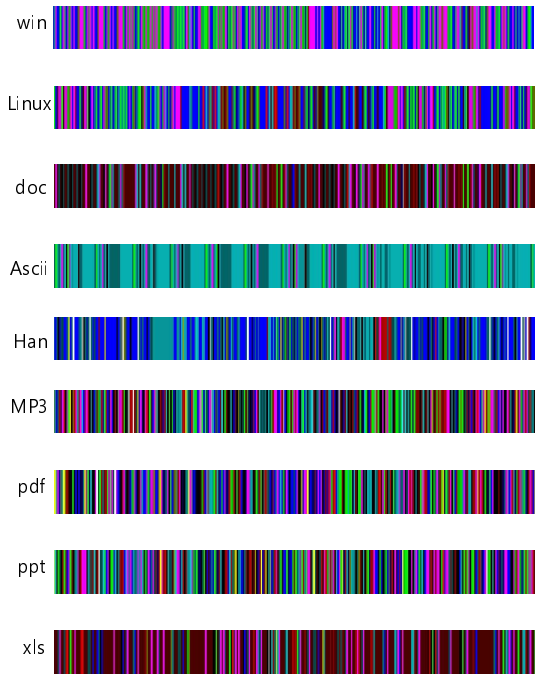


Fig. 4: Instruction spectrum for nine different types of payload.

So if these relations are expressed with a color, a pattern toward the correlation of the executable instruction group can become visible and we can found the executable instruction spectrum. The instruction spectrum analysis method of this thesis could be expressed by the form visually classifying the characteristic of the instruction pattern of executable codes.

First, we classify the several hundreds of IA-32 OP-codes into 109 instruction groups using the libdasm linear disassembler [11], which consist of arithmetic and logical instructions of the integer and floating point, privileged mode and NOP instructions, and so on. Every instruction group is represented by its corresponding color. In order to check the feasibility of basic idea, we disassembled the some kinds of files and we confirmed whether the correlation of the each instruction group is discernable visually.

Figure 4 shows an example of instruction spectrum for nine different types of payload. In Figure 4, a slot indicates an instruction cycle and the color represents its corresponding instruction group. In the Hangul text file (Han), the slots of the blue-green color showing the transfer instruction group are mainly repeated. In case of DOC file (Windows Office), there are many repetitions of the brown and reddish brown slots showing arithmetic and logical calculations respectively. In case of text file (ASCII), there are many repetitions of a bunch of green slots. And in case of non-executable files, there is the tendency that the instruction belonging to the float calculation

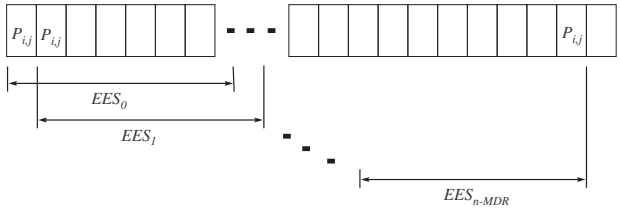


Fig. 5: The expectation value of instruction sequence.

or the others group periodically show up. However, in case of Windows execution PE files, we can see that the instruction sets of control, transfer, stack, logical, arithmetic, and etc are distributed in a rate, and we note that their color spectrums are visually different from ones of the previous three non-executable data files.

As shown in simple example, we can find some different points of instruction spectrum between executable code and non-executable code. So we can say that the basic idea of this thesis could be feasible. However there are some challenging points how to generalize the visual differences in the instruction spectrum and how to automatically determine the executable region.

Now we drive an one-step homogeneous ITPX $P^{(1)}$ as follows,

$$\mathbf{p}^{(1)} = \begin{pmatrix} p_{0,0} & \cdots & p_{0,108} \\ \vdots & \ddots & \vdots \\ p_{108,0} & \cdots & p_{108,108} \end{pmatrix}, \quad (1)$$

where $p_{i,j} = P(X_n = j | X_{n-1} = i), i, j = 0, \ldots, 108, n > 0$ are the transition probabilities from instruction group i to j. In order to get the reference ITPX of executable codes, we scrutinized the sequences of 230,000 instructions in the execution code area (.txt section) of 80 Windows PE(Portable Executable) files stored at windows/system32 folder in Windows System using the libdasm linear disassembler, and then we determined the average transition probabilities from instruction group i to j.
We define two new terminologies as follows,

- *Definition 1*. MDR(Minimum Decision Range) : The number of minimum instructions required for the determination in which the instruction sequence of the executable code exists.
- *Definition 2*. EES(Expectation of Executable Sequence) :

$$EES_n = \Pr\,(y_n, y_{n+1}, \cdots, y_{n+MDR-1} | ITPX), \quad (2)$$

where n is from 1 to $(length(payload) - MDR)$ and y_i's are the observed ith instruction transitions of nth chunk of an IP packet as Figure 5. From the assumption, individual observation y_i's are statistically independent of one another, and we estimate EES by the log function for computational convenience [25] as follows,

$$log\_EES_n = -\sum_{k=n}^{n+MDR-1} \ln p_{i,j}^k, \quad (3)$$

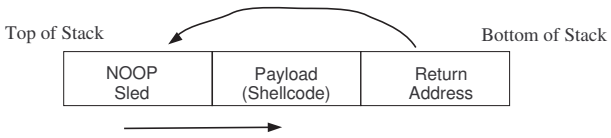


Fig. 6: Typical structure of exploit code using buffer overflow vulnerability.

where $p^k_{i,j}$ is the corresponding $p_{i,j}$ in Eq. 1 of the reference ITPX when the k-th instruction is i and the $(k+1)$-th instruction is j. We note that we choose $MDR = 50$ according to the smallest size of shellcodes of which very short executable codes [36]. Because the optimization of the MDR size is crucial to the shellcode detection, we will consider the optimization problem of MDR in next section.

2) Recognition of Polymorphism: The AVE has the method [13] to be able to detect the polymorphic shellcode in which the disassembly thwarting and shlf-modifying code techniques are used through the static anlysis method. This method performs the disassembly per every byte to detect the *seed* instruction for GetPC so that it is not influenced by the disassembly thwarting technique. Moreover, before the self-modifying code is just operated, the feature of the decyption routine is analyzed and the decyption is detected. The whole idea of this method detects whether the program counter value which the decryption routine stores through the static analysis method is used in accessing the memory or not. In first step, the method finds the *seed* instruction playing the role of storing the program counter value on a stack. As the second, the method detects a register loading the value. The third step is to trace the relation between the register and others. Finally, if the loaded program counter value is used for accessing a memory, the input data is determined to the decryption routine of polymorphic shellcode.

The first step to find the decryption routine is the seed instruction detection for GetPC. The instruction stores the current program counter value on a stack and it is necessary code to find the access address of encrypted code and to use the self-decrypting technique. If an attack already knows the information about the specific register value when the polymorphic shellcode is put on in memory of the remote host, the instruction is unnecessary. However, it is not in an attacker the easy task to predict a situation. Therefore, by using the instruction, in a general way, an attack draws up the decryption routine. As the second step, a description routine loads the program counter value stored in the virtual stack space into the specific register. If the instruction which accesses the memory which is not a stack shows up, it is not the description routine. It is due to the attacker's knowledge limit about the remote host. As the last step, the connection relationship tracing between the other registers and the register in which the program counter value is stored.

3) Recognition of Malicious String: Figure 6 shows the structure of a typical exploit code, which consists of three distinct components – 1) a return address block, 2) a NOOP sled, and 3) the payload. The main purpose of such a construction is that when a function returns following a buffer overflow, the return address block directs execution on to the NOOP sled, which eventually reaches the payload. The AVE has very simple method to detect *NOOP sled* and *return address*. It is similar to pattern matching like signature-based IDS. There are so many NOOP instruction in IA32, so it need to be checked how many similar instructions consecutively appeared. In case of detecting return address, it also check how many 4-byte pattern consecutively show up in the payload. If this kind of simple method is applied to normal network packet, it has very high false rate. However, because this methods are operated with *recognition of executables and polymorphics* technique mutually, its false rate would be very low.

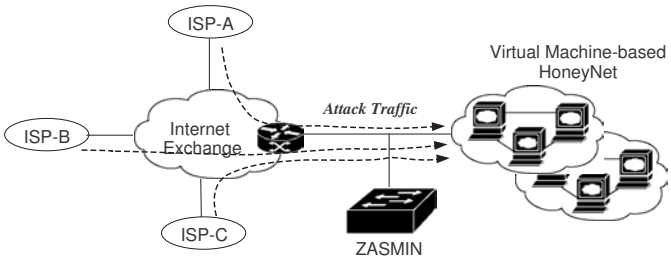


Fig. 7: HoneyNet Testbed in the National Internet Exchange Point (IX).

V. Case Study on the Honeynet Testbed

A. Honeynet Testbed in the IX

In order to evaluate the ZASMIN system in real network environment, we installed it in front of the honeynet, which was constructed in the internet exchange (IX) point as shown in figure 7. This honeynet has been observed by the ASEC[1] in the AhnLab, Inc. It has over 1,500 public IP addresses and various network application services on the virtual machine are launched for inducing attack traffic. It doesn't have any protection appliance like a firewall to temp attack traffic as many as possible. And also, in order that the CERT expert group can analyze the induced traffic, a network monitoring equipment was set up inside honeynet. We want to know as follows through the this experiment.

- Whether the attack information detected in the ZASMIN are real or not.
- The number of the attacks which aren't recognized by CERT out of the attacks detected by the ZASMIN.
- The number of the attack which is recognized by CERT and can generate a signature out of the attacks detected by the ZASMIN.

We have expected to get the system-level false positive rate, and confirmed the capability to detect unknown attacks and the feasibility of the signature generated by the ZASMIN system through the real environmental case study.

[1]AhnLab security emergency response center in the AhnLab, Inc. http://www.ahnlab.com

	Min	Max	Average
Flow Count (per 8 sec. period)	0	1,600	29
Connection Trial Count	0	3,451	26
Connection Success Rate (%)	0	100	32.7

TABLE II: Basic traffic characteristics in the honeynet.

B. Suspicious Traffic & Signature

This experiment had been performed for 48 hours. Table II shows basic traffic characteristics which are monitored on the STD in the ZASMIN system. In the table, the number of flow means the number of the 3-tuple ⟨source IP, protocol, destination port⟩ flow measured for 8 seconds. In average, 29 3-tuple flows appeared and partially, there was the period which doesn't have traffic at all. The connection trial count means the number of SYN packet excluding duplicates. The attempts of the maximum 3,451 and average 26 were monitored for 8 seconds. With regards to TCP connection success rate, average 32.7 % and maximum 100 % were observed for each duration. The traffic volume is a little low, and also these traffic characteristics would be relatively abnormal when it compares with normal traffic monitoring. We consider the reason of this as the characteristics of the honeynet traffic.

As previous mentioned, the STD in the ZASMIN system makes a decision on the suspicious traffic using the dispersion degree of destination IP address, TCP connection trial count, TCP connection success rate and stealth scan trial count. And then, the SEG generates the signature candidates in the suspicious traffic through the content prevalence feature of attack code. As shown in the figure III, total 975 suspicious alerts occurred within 58 flows for 48 hours. The total number of destination port is 11 and the number of source IP addresses is 48. The reason that the number of source IP addresses is less than that of flows is that a source IP address has multiple connections with various TCP destination IP address. As shown in the table III, we can see the most many flows in the destination port 135 (secure shell), and there are many flows in the 445, 139 TCP port. Totally we get 1,105 signatures excluding the duplicate signatures repeatedly generated for 5 minutes. As shown in the table III, there are many signatures at TCP destination port 135 and 445 which are ports for Microsoft server service. When we consider the basic traffic characteristics of the honeynet environment, the volume of traffic is not so much, but the probability to be attack would be very high.

C. Attack Validation

As previous mentioned, the validation functions in the AVE were performed to recognize executables, polymorphics and malicious string. As the result of the attack validation, the number of signature is reduced to 53 attacks at destination port 135, 445 and 9988 as shown table VI. The rest suspicious alerts besides 53 attacks could be any trace of attack scenario, but doesn't have any direct attack evidence like a shellcode of worm. In the table, The number of signature doesn't mean the number of final verdict of the validation but the number of signature about the suspicious attack having at least one more positive result through several validation methods.

dst. port	# of alert	# of flow	# of sig. (unique)	description
21	1	1	–	FTP control
22	17	4	–	Secure Shell
135	309	19	74 (11)	Secure Shell
139	115	10	9 (3)	NETBIOS
445	492	11	1013 (22)	Microsoft-DS
1089	11	3	–	ff-annunc
1433	16	3	3 (2)	Microsoft SQL
2100	1	1	–	Amiganefts/Oracle XDB
2433	5	1	1 (1)	codasrv-se
4899	5	2	–	RAdmin Port
9988	3	3	5 (4)	–

TABLE III: Suspicious traffic information and the number of signature for each port in the honeynet.

dst. port	# of signature	# of unique signature
135	8	6
445	43	11
9988	2	1

TABLE IV: After validation, the number of signature for each destination port.

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 ................
90 90 90 90 90 90 90 90 EB 15 B9 8B E6 13 41 81 ..............A.
F1 39 E6 13 41 5E 80 74 31 FF 85 E2 F9 EB 05 E8 .9..A^.t1.......
E6 FF FF FF B6 5E E1 0E C6 B5 0E C5 89 0E F5 99 .....^..........
28 0E FD 8D 6D C0 85 85 85 D6 D3 0E DA B9 0E D9 (...m...........
BE FD 86 5A D6 0E DE A5 86 5A D6 06 46 81 0E B6 ...Z.....Z..F...
86 72 B6 4C 29 B7 4D 44 44 80 01 45 F0 73 AE 4F .r.L).MDD..E.s.O
F0 6C DD AE 5D 54 6E DB 86 DB A1 86 5A E3 0E 8E .l..]Tn.....Z...
0E DB 99 86 5A 0E 81 0E 86 42 DB DE 7A 65 DB ED ....Z....B..ze..
B6 B7 85 85 ED F2 F6 B7 DA D1 3F 17 EB 81 01 7A ..........?....z
```

Fig. 8: Some part of the payload sample which are used for attack validation.

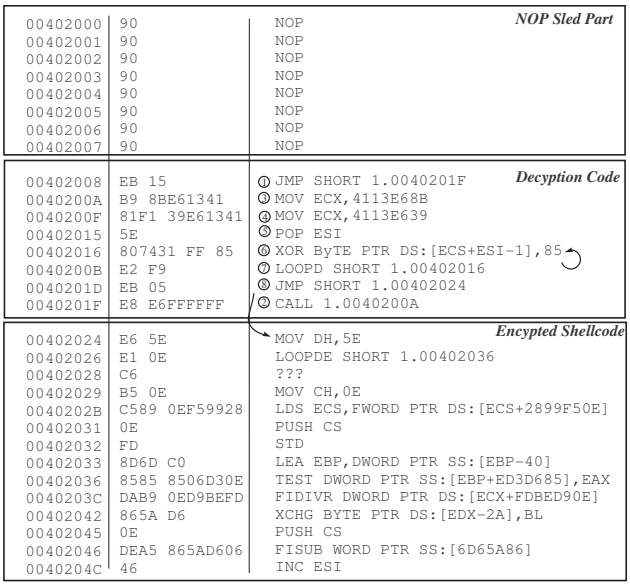


Fig. 9: After disassembling the payload sample, NOOP sled, decrption code of polymorphics and encrypted shellcode are recognised by AVE's validation functionality.

Name of Vulnerability	# of detection
MS06-040	10
(MS05-039)	
MS04-011	16
MS04-007	9
MS04-026	3
MS03-039	3
LSASS	3
Not Attack	1
Not Deterministic	7

TABLE V: The number of attacks for each vulnerability.

We would like to show the details of a representative verification sample out of 53 attacks. Figure 8 show the some part of the payload dumped from the suspicious traffic through the TCP port 135. It is composed of 4 packets and its total length is 1818 bytes, which are reassembled by the preprocessing of the AVE. With regards with the recognition of executables in the AVE, because it doesn't contain the calling function in the shellcode, the call mechanism [12] couldn't detected, but ITPX-based method [15] recognized the executable codes. In the recognition of malicious string, so many NOP codes are detected, but return address couldn't be found out. In this sample, the remarkable point is the recognition of polymorphics. In the figure 8, the red-colored text is the part which are recognized as the decryption codes of the polymorphic shellcode. As shown in the figure 9, after disassembling the payload sample, we can clearly confirm the NOP sled, the decryption codes of polymorphics and encrypted shellcodes. In case of the decryption codes, we can see that just 8 machine instructions can make a effect on decrypting the encrypted shellcode. After the decryption codes following NOP sled codes decode the encrypted codes at address 0x402024 using XOR instruction, instruction pointer jump to the address. Figure 10 shows the shellcode to be executed after decryption. If the payload in the suspicious traffic contains the NOP operation and decryption codes like this, the ZASMIN system can guarantee the suspicious traffic has attack evidence. As the result of the analysis by the CERT team, this suspicious traffic is related to the vulnerability of a name called MS03-039. This is a typical polymorphic worm code using MS03-039 vulnerability, which cause that buffer overrun in RPCSS service could allow code execution. After the validation of attack, the final signature on this attack is released with of the signature syntax of the SNORT as follows. It can be applied to various security appliances with minor change.

ALERT TCP any any → any 135 (msg:"" content:"|57 00 00 00 00 04 5d 88 8a eb 1c c9 11 9f e8|" reference: class-type :0 sigid :186024 revision :0;)

Through this case study, we found several attacks using some vulnerabilities without any well-known signature. Table V show the vulnerabilities used in 53 attacks. In case of not-deterministic, the CERT expert couldn't prove it is an attack or not. Even if these vulnerability was released long time ago, we can confirm that these kinds attacks still exist in the public domain with polymorphic form. As the final decision of the ZASMIN, it determined that 44 alerts are an attack out of 53 attack candidates. The CERT expert group concluded that just one out of 44 attacks is not an attack. It is hard to figure out the exact rate with the small data set like this, the ZASMIN system has the false positive of 2.3 % approximately.

```
00402024    33DB            XOR EBX,EBX
00402026    64:8B43 30      MOV EAX,DWORD PTR FS:[EBX+30]
0040202A    8B40 0C         MOV EAX,DWORD PTR DS:[EAX+C]
0040202D    8B70 1C         MOV ESI,DWORD PTR DS:[EAX+1C]
00402030    AD              LODS DWORD PTR DS:[ESI]
00402031    8B78 08         MOV EDI,DWORD PTR DS:[EAX+8]
00402034    E8 45000000     CALL 1.0040207E
00402039    53              PUSH EBX
0040203A    56              PUSH ESI
0040203B    8B5F 3C         MOV EBX,DWORD PTR DS:[EDI+3C]
0040203E    8B5C3B 78       MOV EBX,DWORD PTR DS:[EBX+EDI+78]
00402042    03DF            ADD EBX,EDI
00402044    53              PUSH EBX
00402045    8B5B 20         MOV EBX,DWORD PTR DS:[EBX+20]
00402048    03DF            ADD EBX,EDI
0040204A    53              PUSH EBX
0040204B    83C3 04         ADD EBX,4
0040204E    8B33            MOV ESI,DWORD PTR DS:[EBX]
00402050    03F7            ADD ESI,EDI
00402052    33C9            XOR ECX,ECX
00402054    AC              LODS BYTE PTR DS:[ESI]
00402055    32C8            XOR CL,AL
00402057    C1C1 05         ROL ECX,5
0040205A    84C0            TEST AL,AL
0040205C  ^ 75 F6           JNZ SHORT 1.00402054
0040205E    2BCA            SUB ECX,EDX
00402060  ^ 75 E9           JNZ SHORT 1.0040204B
00402062    58              POP EAX
```

Fig. 10: After decryption by decryptor, executable shellcode in the payload.

	# of valid attack evidence	# of final attack verdict
ZASMIN System	53	44
CERT Analysis	48	43
False Rate	9.4 %	2.3 %

TABLE VI: False rate of ZASMIN system in this experiment.

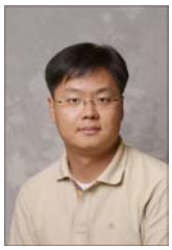
VI. Conclusion

We have introduced the Zeroday-Attack Signature Management Infrastructure(ZASMIN) system for novel network attack detection. This system provides early detection function and validation of attack at the moment the attacks start to spread on the network. After we installed the ZASMIN on real honey-net environment in the internet exchange point (IX), we have analyzed the results of the ZASMIN about detection of unknown attack for two days with CERT expert group. Even if two-day analysis is not enough long to detect various unknown attacks, we could find some attacks without any well-known signature through the case study. Even if these vulnerabilities which the attacks used were released long time ago, these kinds of attacks still exist in the public domain with polymorphic form. Through the this case study, we have convinced that new attack or polymorphic known attack can be detected by the ZASMIN system. It's hard to evaluate the exact system-level false positive rate in the real environment, but we can say that the ZASMIN system has relatively low false rate with this case study. And also, we need to focus on reducing the its false rate as the further study.

References

[1] T. Al-Ibaisi, A. E.-L. Abu-Dalhoum, M. Al-Rawi, M. Alfonseca, and A. Ortega. Network intrusion detection using genetic algorithm to find

best dna signature. *WSEAS Transactions on Systems*, 7(7):589–599, 2008.
[2] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion-detection expert system (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, May 1995.
[3] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection system (NIDES). Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, May 1995.
[4] S. Andersson, A. Clark, and G. Mohay. Network-based buffer overflow detection by exploit code analysis. In *Proceedings of Information Technology Security Conference 2007*, pages 39–53, 2007.
[5] J. Bergeron, M. Debbabi, J. Desharnais, M. Erhioui, and N. Tawbi. Static detection of malicious code in executable programs. *Int. J of Req. Eng.*, 2001.
[6] J. Bergeron, M. Debbabi, M. M. Erhioui, and B. Ktari. Static analysis of binary code to isolate malicious behaviors. In *Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 184–189, 1999.
[7] R. Chinchani and E. van den Berg. A fast static analysis approach to detect exploit code inside network flows. In *Proceedings of RAID*, pages 284–308, 2005.
[8] D. E. Denning and P. G. Neumann. Requirements and model for IDES—A real-time intrusion detection system. Technical report, Computer Science Laboratory, SRI International, 1985.
[9] S. T. Eckmann, G. Vigna, and R. A. Kemmerer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.
[10] S. Hittel. Detection of jump-based ids-evasive noop sleds using snort, May 2002. http://aris.securityfocus.com/rules/020527-Analysis-Jump-NOOP.pdf.
[11] jt. Libdasm, 2006. http://www.klake.org/jt/misc/libdasm-1.4.tar.gz.
[12] D. Kim, Y. Choi, I. Kim, J. Oh, and J. Jang. Function call mechanism based executable code detection for the network security. *SAINT*, 0:62–67, 2008.
[13] D. Kim, I. Kim, J. Oh, and J. Jang. Tracing stored program counter to detect polymorphic shellcode. *IEICE Transactions on Information and Systems*, E91-D(8):2192–2195, 2008.
[14] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of USENIX Security Symposium*, pages 271–286, 2004.
[15] I. Kim, K. Kang, D. Kim, Y. Choi, J. Oh, J. Jang, and K. Han. Executable code recognition in network flows using instruction transition probabilities. *IEICE Transactions on Information and Systems*, E91-D(7):2076–2078, 2008.
[16] C. Kreibich and J. Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (Hotnets II)*, November 2003.
[17] C. Krügel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 201–208, 2002.
[18] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceeding of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143, May 2001.
[19] W. J. Li, K. Wang, S. Stolfo, , and B. Herxog. Fileprints: Identifying file types by n-gram analysis. In *Proceedings of the 6th IEEE Workshop on Information Assurance*, 2006.
[20] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, and P. G. Neuman. A real-time intrusion-detection expert system (IDES). Technical Report Project 6784, CSL, SRI International, Febrary 1992.
[21] M. Mahoney and P. Chan. PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical report, Florida Tech., CS-2001-4, April 2001.
[22] M. Mahoney and P. Chan. Learning models of network traffic for detecting novel attack. Technical report, Florida Tech., CS-2002-8., August 2002.
[23] M. Mahoney and P. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of SIGKDD*, pages 376–385, 2002.
[24] R. Maxion and F. Feather. A case study of ethernet anomalies in a distributed computing environment. *IEEE Transaction on Reliability*, 39(4):433–443, 1990.
[25] I. J. Myung. Tutotial on maximum likelihood estimation. *Journal of Mathmatical Psychology*, 47:90–100, 2003.
[26] J. Newsome, B. Karp, and D. X. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 226–241, 2005.
[27] A. Pasupulati, J. Coit, K. Levitt, S. Wu, S. Li, R. Kuo, , and K. Fan. Buttercup : On network-based detection of polymorphic buffer overflow vulnerabilities. In *Proceedings of 9th IEEE/IFIP NOMS*, pages 235–248, April 2004.
[28] A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networking*, 51(12):3448–3470, 2007.
[29] I. Qualys. The laws of vulnerabilities: Six axioms for understanding risk. In *Qualys White Paper*, 2006.
[30] J. A. Ruiz-Vanoye, O. Diaz-Parra, I. R. Ponce-Medellin, and J. C. Olivares-Roja. Strategic planning for the computer science security. *WSEAS Transactions on Computers*, 7(5):387–396, 2008.
[31] S. Singh, C. Estan, G. Varghese, and S. Savage. The EarlyBird system for realtime detection of unknown worms. Technical Report CS2003-0761, UC San Diego, August 2003.
[32] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI*, pages 45–60, 2004.
[33] SNORT. The open source network intrusion detection system, 2002. http://www.snort.org/.
[34] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.
[35] C. Taylor and J. Alves-Foss. Nate: Network analysis of anomalous traffic events, a low-cost approach. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 89–96, 2001.
[36] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of RAID*, pages 274–291, 2002.
[37] H. Wang, C. Guo, D. Simon, and A. Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of ACM SIGCOMM*, pages 194–204, 2004.
[38] K. Wang, G. F. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of RAID*, pages 227–246, 2005.
[39] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of RAID*, pages 203–222, 2004.
[40] X. Wang, C.-C. Pan, P. Liu, and S. Zhu. SigFree: a signature-free buffer overflow attack blocker. In *Proceedings of the 15th conference on USENIX Security Symposium*, 2006.
[41] K.-M. Yu, M.-F. Wu, and W.-T. Wong. Protocol-based classification for intrusion detection. *WSEAS Transactions on Computer Research*, 3(3):135–141, 2008.



Ikkyun Kim , received the B.C.E. and M.S.C.E. degrees in computer engineering from the Kyungpook National University, Daegue, South Korea, in 1994 and 1996, respectively, and is currently working toward the Ph.D. degree. He worked at ETRI (Electronics and Telecommunications Research Institute) from 1996 to 1999. During 2000-2001, he stayed in Paxcomm, Ltd, as a Research staff. From 2001, he works with the Security Gateway Team, ETRI, Daejeon, Korea. He has developed several network-based intrusion detection systems and is currently working on new anomaly detection method against zero-day attacks for netwrok security. His research interests include DDoS protection mechanism, high-speed network protection system and the design of network processor architecture for network security appliance.

Daewon Kim , received the BS degree in electrical engineering from Kyungpook National University and the MS degree in electrical engineering from Korea Advanced Institute of Science and Technology, South Korea, in 2003 and 2005, respectively. Since 2005, he has been with Electronics and Telecommunications Research Institute, Daejeon, South Korea, where he is currently a researcher with Information Security Research Division. His current research topics are the detection methods against zero-day/unknown attacks with ZASMIN project. His recent interests are focusing on the polymorphic exploits detection and behavior-based malware analysis.

Koohong Kang , received the B.Sc. and M.Sc. degrees in Electronics Engineering from Kyungpook National University and Chungnam National University, South Korea, in 1985 and 1990, respectively, and Ph.D. degree in Computer Science and Engineering from Pohang University of Science and Technology (POSTECH), South Korea, in 1998. Dr. Kang is currently an associate professor at the Dept. of Information and Communications Engineering, Seowon University, South Korea. From 1985 to 1993 and from 1999 to 2000, he was a member of technical staff in Electronics and Telecommunications Research Institute (ETRI), South Korea. Dr. Kang has developed several switching systems for high-speed router, ATM, and telephony, and currently working on scheduling and security for wireless multi-hop TDMA networks. His research interests include zero-day attack detection, wireless multi-hop scheduling, and performance evaluation.

Yangseo 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 in the doctorate course of Computer Engineering from Chung-nam National University and is currently working on new anomaly detection method against zero-day attacks for netwrok security in ETRI. His research interests include DDoS protection mechanism, high-speed network protection system, malware analysis and forensics.

Jintae Oh , received the B.E.E. and M.S.E.E. degrees in electronic engineering from the Kyungpook National University, Daegue, South Korea, in 1990 and 1992, respectively, and is currently working toward the Ph.D. degree. He worked at ETRI (Electronics and Telecommunications Research Institute) from 1992 to 1998. During 1998-1999, he stayed in MinMax Tech , USA, as a Research staff. He served as a Director in Engedi Networks, USA, during 1999-2001. He was both Co-founder and CTO Vice President in Winnow Tech. USA during 2001-2003. From 2003, he works with the Security Gateway Team, ETRI, Daejeon, Korea. He has developed several network-based intrusion detection systems and is currently working on new anomaly detection method against zero-day attacks for netwrok security. His research interests include DDoS protection mechanism and FPGA-based high speed pattern matching algorithm for intrusion detection system.

Jongsoo Jang , received the B.E.E. and M.S.E.E. degrees in electronic engineering from the Kyungpook National University, Daegue, South Korea, in 1984 and 1986, respectively, and the Ph.D degree in computer engineering from the Chungpook National University, Chengjoo, South Korea, in 2000. Since 1989, he has been working with ETRI, Daejeon, Korea and now is the Director of Applied Security Group. Since January 2008, he has been a Vice-President of KIISC(Korea Institute of Information Security and Cryptology) He has developed several network-based intrusion detection systems and is currently working on new anomaly detection method against zero-day attacks for netwrok security. His research interests include network security management and FPGA-based high speed pattern matching algorithm for intrusion detection system.