IEICE TRANS. INF. & SYST., VOL.E96–D, NO.8 AUGUST 2013

PAPER

PC Worm Detection System Based on the Correlation between User Interactions and Comprehensive Network Behaviors

Jeongseok SEO[†a)], ***Student Member***, **Sungdeok CHA**[††], **Bin ZHU**[†††], ***and*** **Doohwan BAE**[†], ***Nonmembers***

SUMMARY Anomaly-based worm detection is a complement to existing signature-based worm detectors. It detects unknown worms and fills the gap between when a worm is propagated and when a signature is generated and downloaded to a signature-based worm detector. A major obstacle for its deployment to personal computers (PCs) is its high false positive alarms since a typical PC user lacks the skill to handle exceptions flagged by a detector without much knowledge of computers. In this paper, we exploit the feature of personal computers in which the user interacts with many running programs and the features combining various network characteristics. The model of a program's network behaviors is conditioned on the human interactions with the program. Our scheme automates detection of unknown worms with dramatically reduced false positive alarms while not compromising low false negatives, as proved by our experimental results from an implementation on Windows-based PCs to detect real world worms.

key words: *worm detection, personal computer security, Internet worm*

1. Introduction

Ever since the "Morris worm" took the Internet down in 1988, many worms (e.g., CodeRed, Nimda, Slammer, Blaster, Sasser, Witty) [1]–[5] continue to inflict significant damage. CodeRed v2, in 2001, infected nearly 400,000 hosts in just 14 hours and caused damage estimated at $2.6 billion [1]. Security experts fear that future worms would spread faster [6]–[8], be more lethal, and become stealthier than the past generations. While corporations often install several layers of defensive mechanisms (e.g., intrusion detection systems, honeypots, etc.) to protect themselves against the threat of Internet worms, "ordinary" PC users who use Internet to perform routine but essential tasks (e.g., e-banking, blogging, etc.) have insufficient protection in place. They rarely possess technical knowledge on PC security, yet many of their computers are always connected to the Internet and store personal or sensitive information (e.g., electronic banking statements, e-trading keys, etc.). Even the latest anti-virus software and firewalls are ineffective in protecting a PC when a new worm is launched until the worm signatures are updated.

Manuscript received November 26, 2012.
Manuscript revised April 13, 2013.

[†]The authors are with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea.
[††]The author is with the Department of Information and Communications, Korea University, Seoul, Republic of Korea.
[†††]The author is with the Microsoft Research Asia, Beijing, P.R. China.
a) E-mail: jsseo@se.kaist.ac.kr

DOI: 10.1587/transinf.E96.D.1716

PC-based Worm Detection System, PC-WDS in short, has the following design goals:

- Fully automated operation: PC-WDS must be capable of detecting previously unknown worms even when no signatures are available, and its operation must be fully automated. As Windows operating systems dominate the PC market, we demonstrate the effectiveness of the proposed design using Windows-based real Internet worms.
- Low false positives: If "normal" programs frequently used by users are incorrectly labeled as worms, users would distrust PC-WDS and choose to uninstall it.
- Small overhead: If users notice their PCs running noticeably slow after installing PC-WDS, they would probably reject the system.

There exist diverse approaches to detect malware. For example, Spitzner [9] and Provos [10] deployed multiple honeypots to detect malicious behavior by integrating information collected by various sensors. Dagon et al. [11] detected local scanning worms using multiple honeypots. While clearly effective, such approaches do not adequately protect PCs because typical users rarely possess the necessary resources to deploy honeypots or technical knowledge to manage them. Monitoring network traffic can also detect worm propagation [12]. A surge of packets to unreachable destinations is clearly abnormal, and Internet Storm Center [13] analyzes logs from many Internet-related organizations. Riordan et al. [14] developed a system using multiple monitors and virtual honeypot systems installed at various IBM-associated companies. EarlyBird system [15] automatically generates potential worm signatures by inspecting traffic patterns and relationships between sources and destinations addresses. Xia et al. [16] demonstrated how to detect worms by analyzing packets with unused destination addresses.

Unfortunately, research on worm detection on PC is still immature. BINDER [17], based on the intuition that worm-initiated network connections rarely have the triggering user inputs (e.g., mouse clicks and keyboard inputs), uses time differences between network requests and the preceding user events as detection feature. It assumes that all user-initiated network requests are normal, and the whitelist allows users to designate "legitimate" daemons to reduce false positives. While useful, BINDER's detection feature is neither powerful nor flexible enough to detect new worms. Attackers may attempt to evade detection by faking

user events or infecting normal programs with worm code. Should a user forget to include daemons in the whitelist, false positives may occur. On the contrary, if worms somehow manage themselves to be included in the whitelist, BINDER system would become ineffective. More importantly, typical PC users may find BINDER difficult to use.

In this paper, we describe the design of a PC-WDS which extends sound ideas proposed in the BINDER system with sophisticated features to improve detection accuracy while reducing false alarms. To demonstrate the effectiveness of PC-WDS, we launched real Internet worms on a system running a newly installed Windows operating system. We used worms collected by the China Honeypot project [18], and they played the role of previously unknown worms in the experimental setting. Experimental results show that PC-WDS is highly effective in detecting new worms.

The rest of our paper is organized as follows. Section 2 describes PC-WDS architecture with emphasis on how various features work together to achieve a powerful and flexible worm detection environment. Section 3 shows the implementation detail of PC-WDS including making normal behavior profile and detecting Internet worms. Section 4 explains how effectiveness of PC-WDS is evaluated in a realistic and representative setting. Section 5 concludes the paper.

2. PC-WDS Architecture and Worm Detection Features

Scanning of vulnerable hosts, the most apparent characteristic of Internet worms is generation of a large number of network connection requests, and many timeouts and "destination unreachable" ICMP messages are likely to occur. Brute-force application of such feature causes frequent false positives because "normal" applications may also exhibit similar behavior when the network is slow or the links are broken. For example, Fig. 1 illustrates that the Blaster worm and the Internet Explorer (IE) exhibit the failed network connections and the user interactions such as mouse clicks and keyboard inputs. While both behavior might be similar in terms of failed network connections, existence of user triggered inputs separate the two.

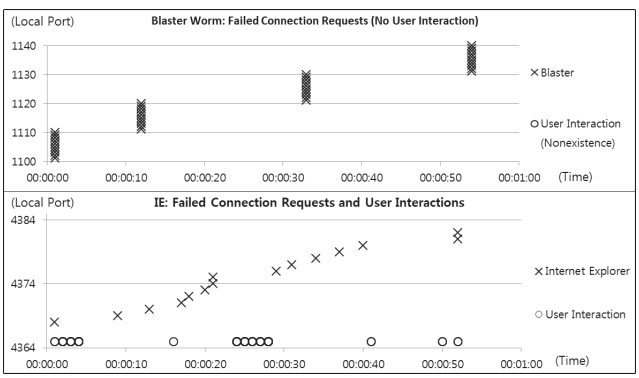


Fig. 1 Failed network connection requests caused by Blaster worm and Internet Explorer (IE).

Analysis of network activities in the context of preceding user interactions is the key idea behind the worm detection technique used by BINDER and PC-WDS. In our approach, we monitor Window's SendInput() API to determine if events had really been generated by "genuine" hardware interrupts [19]. This approach is effective if a worm runs as a user level process. If the worm can somehow run in the kernel mode, there is little PC-WDS can do because the "bad guy" has already full control over the PC and can simply terminate any process of its choice including PC-WDS. We assume that modern operating systems would provide adequate, if still imperfect, process protection mechanism.

2.1 PC-WDS Architecture

The PC-WDS architecture, shown in Fig. 2, consists of the following modules: data collector (event monitor), user interaction analyzer, destination address analyzer, network connection analyzer, port creation analyzer, and detection engine. Data collector monitors user interaction (keyboard/mouse) events, network packet events and Windows process-related events. The user, network and system related events are transmitted to 4 core data analyzers to build corresponding profiles, which are integrated and flagged by detection engine.

2.2 Data Collection

Our monitoring agents collect the following activities: (1) keyboard and mouse click events; (2) network communication events; (3) local port creations or destructions events. To protect user's privacy, our system records process ID, event type, and timestamp but not the "content" (e.g., which key was pressed). Data on network communications include the source and destination IPs, ports, traffic direction (e.g., incoming or outgoing) and timestamp. Likewise, when local ports are created or destroyed, process ID, process image name, port number, action type (e.g., creation or destruc-

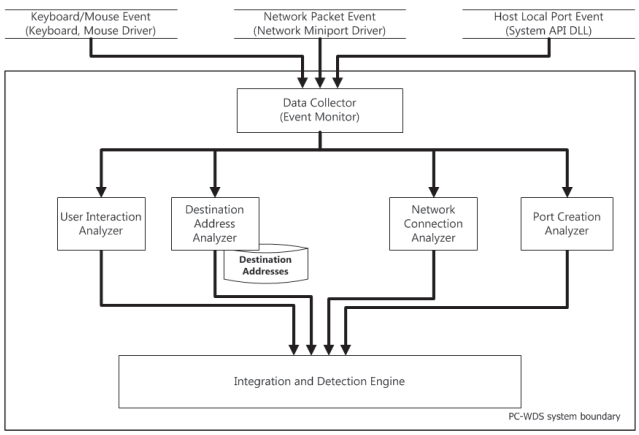


Fig. 2 PC-WDS architecture.

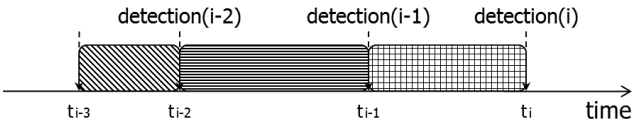


Fig. 3 Time sliding windows for detection (each time windows are randomly chosen in between lower bound and upper bound).

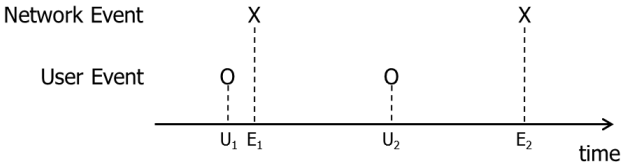


Fig. 4 Likelihood of machine-triggered activity for keyboard or mouse click events triggering network events ($LM(E_2) > LM(E_1)$).

tion), and timestamp are recorded. Note that process image name, as opposed to the process ID, is recorded because multiple instances of the same program (e.g., IE) might be running simultaneously.

In implementation detail, data collector has 3 agents based on different sources from user, network and Windows process behaviors. User behavior monitoring agent uses SetWindowsHookEx() API to get low-level keyboard and mouse input events [20]. Network agent records TCP/UDP connections with Microsoft Research user mode capture miniport driver, and AllocateAndGetTcpExTableFromStack()/AllocateAndGetUdpExTableFromStack() APIs in IPHLPAPI.DLL, understanding that which processes are using corresponding TCP/UDP connections [20]. In addition, our process monitoring agent uses many system APIs in KERNEL32.DLL and PSAPI.DLL, providing plentiful process status information [20]. Data collector gathers all information from 3 agents and feeds them to data analyzers.

Even if we developed very effective worm detectors, malware writers eventually learn about the implementations of our system and can evade our detector. To avoid detection, the malware programs can remain dormant occasionally over a period of monitoring time and remain "patient enough" to limit scanning activities in such a manner that the anomaly score would never exceed the threshold value. To prevent this failure of worm detection, we randomly choose monitoring time intervals as shown in Fig. 3. If the monitoring time interval is unpredictable, then it will be difficult for the worms to evade detection. To make effective detection intervals without degraded accuracy, every randomly-chosen interval is in between lower bound and upper bound as a detection time window.

2.3 Basic Detection Attribute 1: User Interaction Analysis

Output from the data collector is used to correlate network events with user's interactions. PC-WDS uses time difference to estimate the likelihood that a user's interaction really triggered network connections. Likelihood of machine-triggered activity, $LM(E)$, is computed as follows:

$$LM(E) = \begin{cases} \frac{T_E - T_U}{TD_{MAX}} & \text{, if the most recent user event exists} \\ 1 & \text{, otherwise or } LM(E) > 1 \end{cases}$$

where T_U refers to the timestamp of the most recent user event preceding network event E, and T_E means the timestamp of event E. TD_{MAX} is the maximum time difference which has been seen before between T_U and T_E. In other words, $LM(E)$ is assigned 1, which means that the network event is assumed have been generated without a triggering user event. Figure 4 shows that the value of $LM(E_2)$ is larger than that of $LM(E_1)$.

TD_{MAX} is the maximum delay considered possible between a user event and triggered network events ideally. If TD_{MAX} is too small, false positives would increase because PC-WDS have difficulty of differentiating "valid" and user-initiated network requests from worm scanning activities. TD_{MAX}, while not compromising low false positives, continues to grow from lower bound of detection time window initially when bigger time difference will have been met than existing one. When a user is busy at keyboard typing or clicking, time difference would be pretty small. However, PC-WDS must be smart enough to understand that a normal application (e.g., MSN messenger) might continually generate network connections in the absence of user inputs. If a user leaves messenger application running while away from the terminal, as it is frequently the case, there will be a series of network connections generated to check if friends have just signed on, etc. Such behavior appears, when analyzed using time difference feature in a naive manner, like that of a worm. In the PC-WDS design, like the BINDER system, we assume that a network event is less likely to have been triggered by user input if the time difference increases. Such analysis is not a boolean decision in nature, and lower probability value of user-triggered is assigned when time difference increases. To enable fully automated detection practically while not compromising low false positives, we obtained data on all the time differences between a network event and the most recent and preceding user event observed by all the "normal" applications we had tested in our experimental environment and explained in experimental setting section. Statistical analysis, performed with 99.97% confidence level, returned 1,354 seconds to be exact, as not to exceed the maximum TD_{MAX} value. In other words, TD_{MAX} value is keep growing during run-time operation of PC-WDS, when larger value happens than existing one, but TD_{MAX} value is not to exceed the maximum 1,354 preventing intolerable big change of the likelihood. Therefore if the time differences are over 1, $LM(E)$ is assigned 1 by our definition.

2.4 Basic Detection Attribute 2: Destination Address Analysis

PC-WDS must accurately distinguish Internet worms from "legitimate" system software (e.g., system daemons, self-update software, or patch downloads) that does not require explicit user inputs. In Microsoft Windows 2000 and XP, ex-

amples include System, lsass.exe, services.exe, spoolsv.exe and svchost.exe. Fortunately, behavior of typical daemons and update software are different enough from that of worms in that the former tends to establish network connections with a relatively small number of predefined hosts. In addition, connections are attempted periodically, and the list of remote hosts changes infrequently.

Destination address analyzer computes the degree of destination address diversity based on such intuition. PC-WDS maintains the list of unique addresses each program image most recently communicated with. Addresses are maintained in the LRU (least recently used) manner. Destination address analyzer computes Ratio of New Destination Address ($RT(E_{NDA})$, for short) as follows:

$$RT(E_{NDA}) = \frac{|DA_{New}|}{\left|DA_{Profile}\right|}$$

where E_{NDA} refers to event E whose destination address is never seen before, Destination Address = {Destination IP, Destination Port}, $DA_{Profile}$ = {Destination Address |Histroy of Destination Address} and DA_{New} = {Destination Address |Destination Address $\notin DA_{Profile}$}.

Generally if $|DA_{Profile}|$, the number of distinct destination addresses PC-WDS maintains for each program, is too small, PC-WDS would incorrectly interpret repeated requests to destination addresses as new, and normal programs might be incorrectly classified as a worm. This is particularly the case with P2P software. In PC-WDS design, profile size must be large enough not to classify P2P software as worms but small enough to keep system overhead (e.g., data structure) at a reasonable level. For the practical operation of PC-WDS, profile storage limit 10,000 was adequately chosen because: (1) P2P software (e.g., emule.exe and utorrent.exe) communicates with a large number of hosts simultaneously, and (2) sufficient safety margins were added to reduce false positives while not compromising detection accuracy. The bigger profiles we use, the more correctly the attempted destination address patterns might be calculated. However, with larger profile storage size than 10,000, we found that we didn't get the more accurate detection rate. Moreover, if worms try to avoid detection by scanning less than 10,000 hosts, security threat posed by such worms would be practically insignificant.

If all of the attempted destination addresses are new for the randomly given duration, ratio of new destination address would be 1. Figure 5 illustrates that worms and other applications had big difference in the diversity of destination addresses. While clearly useful, this feature alone is not as effective as it appears. For example, "sneaky" worms may attempt to avoid detection by periodically revisiting destination addresses with the sole purpose of defeating PC-WDS. Therefore, PC-WDS uses more sophisticated features to improve detection accuracy while reducing false alarms.

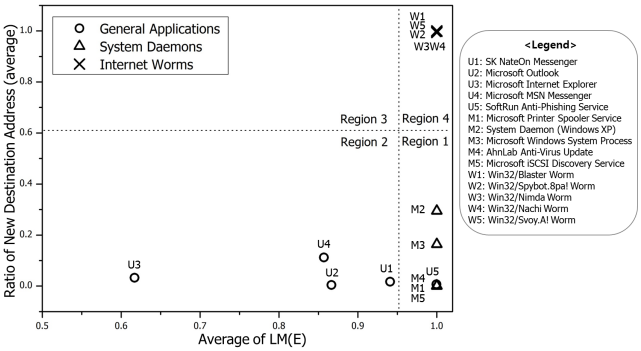


Fig. 5 Diversity of destination addresses.

2.5 Basic Detection Attribute 3: Network Connection Analysis

Network connection patterns determine effectively whether there are worms running on PC or not. PC-WDS analyzes failed network connections because worms are much more likely to generate a large number of unsuccessful network connections. Computation of anomaly score is repeated at every random time interval: 1) average likelihood of machine-triggered activity for every event of failed network connections and 2) ratio of failed network connections for each sliding time window:

1) $AVG(LM(E_F))$ = Average of $LM(E_F)$
2) $RT(E_F)$ = Ratio of Failed Network Connection

where E_F refers to event E whose network connection trial is failed.

Rationale is that (1) much failed request must contribute enough to ratio of failed network connections; and (2) non-user interaction behaviors used also contribute enough to average $LM(E_F)$. However longer monitoring time intervals may get more events than shorter one. To prevent this unfairness, we use an average value.

2.6 Basic Detection Attribute 4: Port Creation Analysis

The other feature involves patterns on local port creation and the likelihood that they have been triggered by user events. Although Blaster worm and Internet Explorer created similar number of local ports (see Fig. 1), anomaly scores must show big difference. PC-WDS uses the following formula: 1) average likelihood of machine-triggered activity for every event of port creations and 2) average number of port creations for each sliding time window:

1) $AVG(LM(E_P))$ = Average of $LM(E_P)$
2) $AVG(N(E_P))$ = Average Number of Port Creation

where E_P refers to event E whose network port opens to make a connection. Average number can be calculated by dividing each time interval of sliding windows to prevent unfairly biased anomaly score in long time window.

A large number of unsuccessful network connections and port creations are apparent characteristic of Internet worms above mentioned in Sects. 2.5 and 2.6. However, deployment of sole feature is not effective because it's too simple and can be evaded easily. In experimental results section, we will evaluate the deployment of sole feature. Therefore PC-WDS uses comprehensive approach to combine user interactions and network based detection features.

3. Implementation

3.1 Noise Filtering

To reduce the effect of outliers such as sudden increases and decreases of anomaly scores, a moving average filter is applied based on detection sliding windows. The moving average filter removes high-frequency noise from the analysis, resulting in more smooth patterns. While calculating the average of its neighboring detection windows in size $2k + 1$, each anomaly score of window in detection (i) is calculated into $AS_f(i)$ as follows:

$$AS_f(i) = \frac{AS(i-k) + \cdots + AS(i) + \cdots + AS(i+k)}{2k+1}$$

The window size determines the smoothness of the anomaly score changes. For examples, the larger k is, the smoother the curve is, however we may lose the sensitive detection of Internet worms. If the small k we use, the detector is sensitive about anomaly score fluctuations but likely to result in false positives. The impact of window size k associated with the detection accuracy will be evaluated in experimental results section.

3.2 Learning Normal Behavior Profile

Support Vector Machine (SVM, for short) has been successfully used in many applications including various research areas as a maximal-margin classifier [21], [22]. Some anomaly detection systems have been based on SVM [23]–[25], primarily because SVM has shown outstanding performance for various research experiments. Therefore we have carried out researches to make normal behavior profile while on decreasing the number of false alarms of worm detection by using comprehensive features based on SVM.

Scholkopf et al. [26] proposed one-class SVM, which only uses training examples from one-class, instead of multiple classes. In training phase, the one-class SVM first maps training data into a high dimensional feature space via a kernel function, and finds the maximal margin hyper-plane that best optimizes a radius of the training data sphere in a high dimensional feature space. One-class SVM approach uses unsupervised learning of normal behaviors for detecting outliers as Internet worms; that means one-class SVM does not require labeled data for unknown worms but only requires normal application behaviors that a user practically installed in his/her desktop. It is almost impossible to get all labeled data of future and unknown Internet worms.

In this work we use the LIBSVM 3.12 [27] for our experiments. The following 6 features as an input feature vector of SVM are used for normal behavior profile of PC-WDS:

$$\begin{aligned} < \quad & AS_f(AVG(LM(E_F))), AS_f(RT(E_F)), \\ & AS_f(AVG(LM(E_P))), AS_f(AVG(N(E_P))), \\ & AS_f(AVG(LM(E_{NDA}))), AS_f(RT(E_{NDA})) \quad > \end{aligned} \tag{1}$$

Normal behavior profile includes likelihood of machine-triggered for failed network connections, port creations and new destination addresses, in which a moving average filter scheme is applied and average values are used. In addition, ratio of failed network connections and new destination addresses and average number of port creations with a moving average filter are profiled. As we already discussed in detection attributes, all scores are increasing to contribute enough to generate anomaly scores for Internet worms. In experimental evaluation section, we will discuss more details about training data for normal behavior profile and test data to evaluate our worm detection system.

3.3 Detecting Internet Worms

To detect Internet worms in PC-WDS, we use one-class SVM. SVM Learner can make a normal behavior profile of legitimate applications with comprehensive features in Eq. (1). This normal profile has no prior knowledge (e.g., signature) of worms and only behaviors of normal applications. SVM Classifier can detect outliers as Internet worm behaviors with this normal profile. Figure 6 shows the implementation detail of our integration and detection engine in PC-WDS architecture. The voting engine decides if consecutive results of SVM Classifier are to be considered as being anomalous. If the number of consecutive abnormal results exceeds threshold value, the current program is considered as Internet worm finally. The voting engine is designed to reduce false alarms due to the marginal error of SVM one-class model robustness to make the best generalization. To evaluate the effectiveness of our worm detec-

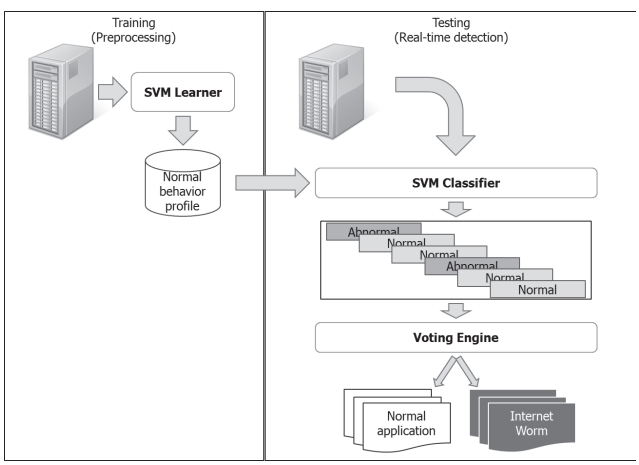


Fig. 6 Implementation detail of integration and detection engine in PC-WDS architecture.

tion features and generic PC-WDS performance, we use a threshold value of 1 as the number of consecutive abnormal results in our experimental evaluation; that means experimental result shows the direct result of SVM classifier.

4. Experimental Evaluation of PC-WDS

4.1 Experimental Setting

To measure the effectiveness of PC-WDS, we collected data for 28 days from 10 "ordinary" PC users who use Microsoft Windows on Pentium 4 or higher processors equipped with main memory ranging from 512 MB to 4 GB. Other than installing PC-WDS, no parameter settings or tunings were applied. There were no constraints enforced on the participants. For example, users installed new software during experiment if needed. Table 1 illustrates the number of applications, ranging from 34 to 65, each participant used. Because some applications (e.g., Outlook, IE, Messengers, etc.) are popular with many users, the number of applications is 152.

When performing an experiment on worm detection, we used PCs that had newly installed therefore worm-free Windows XP operating system. Each worm was released for 24 hours, and data collection was performed. We repeated the process of installing a clean Windows XP and launching a worm. It must be emphasized that PC-WDS has no prior knowledge of worms used in the experiment. It detects presence of worms based solely on observed normal program behaviors. In our experiment, 23 real world worms are exploited to prove our detection capability: Win32/Adload, Win32/Agobot, Win32/Blaster, Win32/Bobax, Win32/Dabber.B, Win32/Deadhat, Win32/Doomjuice, Win32/Forbot, Win32/IRCBot.Gen, Win32/IRCBot.worm.variant, Win32/Korgo, Win32/Nachi, Win32/Nanspy, Win32/Nimda, Win32/Parite, Win32/Prex.R, Win32/SdBot, Win32/Shorty, Win32/Spybot.8pa!, Win32/Svoy.A!, Win32/Tenga, Win32/Valla, Win32/Xema.

As it is impractical to describe the analysis of the behaviors of all 152 applications against all the worms in this paper, we choose five most frequently used applications as well as five best-known Windows-based worms as representatives to interpret and evaluate our worm detection system, as shown in Table 2. In addition, we choose five system daemons and automatic update programs as representative samples. We denote each as U1 through U5 (user applications), W1 through W5 (worms), and M1 through M5 (updates or daemons). We include 5 Windows-based worms of representative samples based on their scanning strategy [16], [28] as shown in Table 3. However we cannot exploit Linux worm because it's not working on Windows system. Likewise, because of the absence of real Windows-based worm, some of scanning strategy cannot be tested.

To make the SVM profile of legitimate applications which can effectively distinguish Internet worms from normal applications, SVM Learner developed profiles based on several different hours of normal operation for each user. After training, SVM Classifier tests all remaining behaviors of applications and real Internet worms running on PCs, and triggers outliers as Internet worm. The result based on each normal profile can be shown in experimental results section.

4.2 Experimental Results

When failed network connection patterns are solely examined in computing anomaly score without noise filter scheme, PC-WDS accurately detected three worms (W2, W4, and W5) (see Fig. 7). Two worms, Blaster (W1) and Nimda (W3), successfully evaded detection because they experienced relatively low rate of failed network connections. As expected, all user applications demonstrated high likelihood of user-triggered interactions. System daemons M2 and M5 resulted in receiving higher anomaly scores than user applications or other daemons. System (M2) issued many UDP data packets to local sub-network (netbios services) without receiving any response. Likewise, Microsoft iSCSI discovery service, M5, periodically made failed network connections during the experiment. Therefore, worm detection based solely on failed network connection is inaccurate.

Table 1 Number of "normal" applications used by 10 participants.

	Number of different applications
User 1	55
User 2	46
User 3	35
User 4	42
User 5	65
User 6	46
User 7	38
User 8	45
User 9	34
User 10	42
Total User	152

Table 2 Five most popular entries in each category.

User Applications		System Daemons & Automatic Update		Worms running on Windows	
nateonmain.exe	**U1**	spoolsv.exe	**M1**	Win32/Blaster	**W1**
outlook.exe	**U2**	System	**M2**	Win32/Spybot.8pa!	**W2**
iexplore.exe	**U3**	svchost.exe	**M3**	Win32/Nimda	**W3**
msnmsgr.exe	**U4**	acaas.exe	**M4**	Win32/Nachi	**W4**
nophishing.exe	**U5**	iscsiexe.exe	**M5**	Win32/Svoy.A!	**W5**

Table 3 Different scanning strategies and representative samples of worms.

Victim discovery method	Internet Worm	
Sequential	Win32/Blaster.Worm	W1
	Win32/Svoy.A!Worm	W5
Random	Win32/Spybot.8pa!Worm	W2
Local (subnet)	Win32/Nimda.Worm	W3
	Win32/Nachi.Worm	W4
	Win32/Svoy.A!Worm	W5
Selective random	Linux/Slapper.Worm	
Routable, Divide-conquer, Hybrid or Permutation	No such worms reported yet	

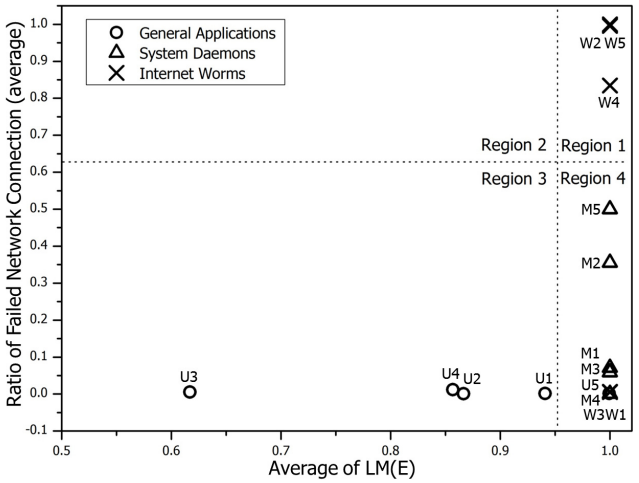


Fig. 7 Classification of 15 programs by the network connection patterns (without noise filter scheme).

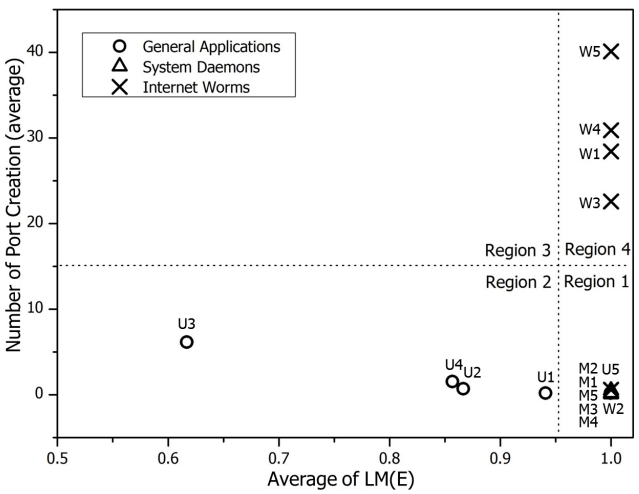


Fig. 8 Classification of 15 programs by the average number of port creation (without noise filter scheme).

Worm detection based on the port creation pattern is also imperfect. Figure 8 indicates that PC-WDS missed Spybot.8pa! worm, labeled W2, which created only 0.8 ports every minutes to avoid detection at the cost of slower scanning and infection. This is another reason that more comprehensive features need to be applied in a flexible manner to accurately detect unknown worms.

Figure 9 shows the effect of noise filtering applied. As the window size becomes growing, the curves of anomaly score have been smoothly changed. Noise filter scheme removes such sudden spikes of anomaly score effectively and reduces false positive alarms for worm detection.

To evaluate the detection performance with a noise filtering scheme, Fig. 10 shows total pinpoint accuracy of 28 days operation of 152 normal applications and 24 hours operation of 23 Internet worms. In our experiment, one-class SVM uses radial basis kernel function and gamma and nu parameters are configured based on 10-fold cross validation with LIBSVM 3.12 [27]. As the window size K becomes growing, generally total pinpoint accuracy of PC-WDS is increasing. With $K = 3$, the pinpoint accuracy exceeds 99.9%, which reasons that low false positive alarms are shown. When K is increasing from 4, pinpoint accuracy slightly increases but it's already exceeding 99.9%. With the large K, learning data may be too much abstracted and difficult to make separable between normal and worm behaviors. Additionally with the large K, SVM marginal errors are also growing.

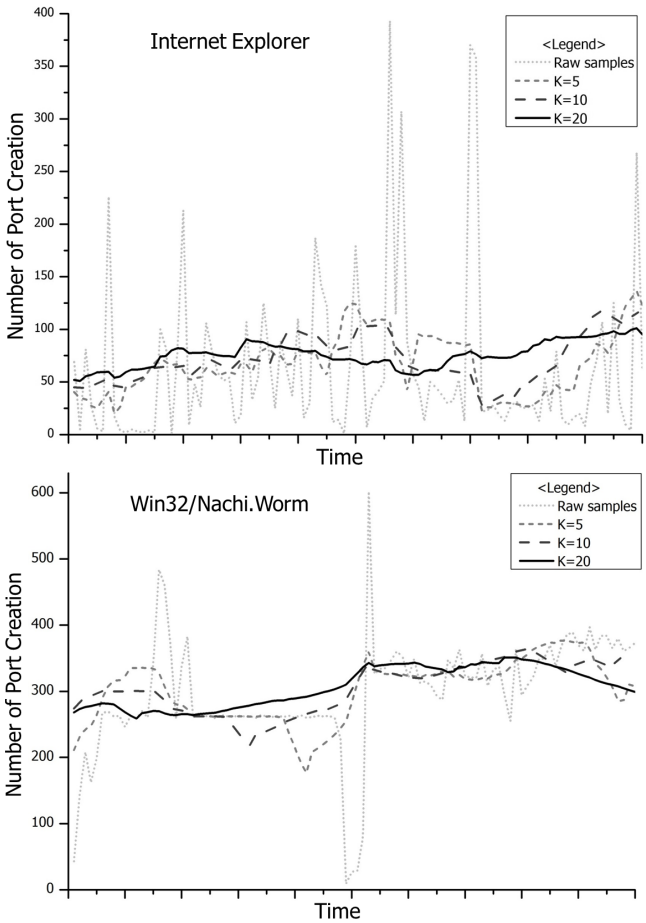


Fig. 9 Anomaly score (number of port creation) trend with and without noise filtering applied (Internet Explorer and Win32/Nachi worm).

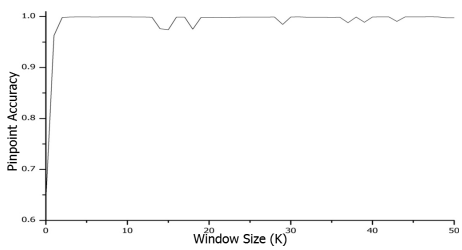


Fig. 10 Noise filter effect.

Figure 11 shows that 3 scatter matrix diagrams of whole training and testing data including 152 normal applications and 23 Internet worms, when the nose filter pa-

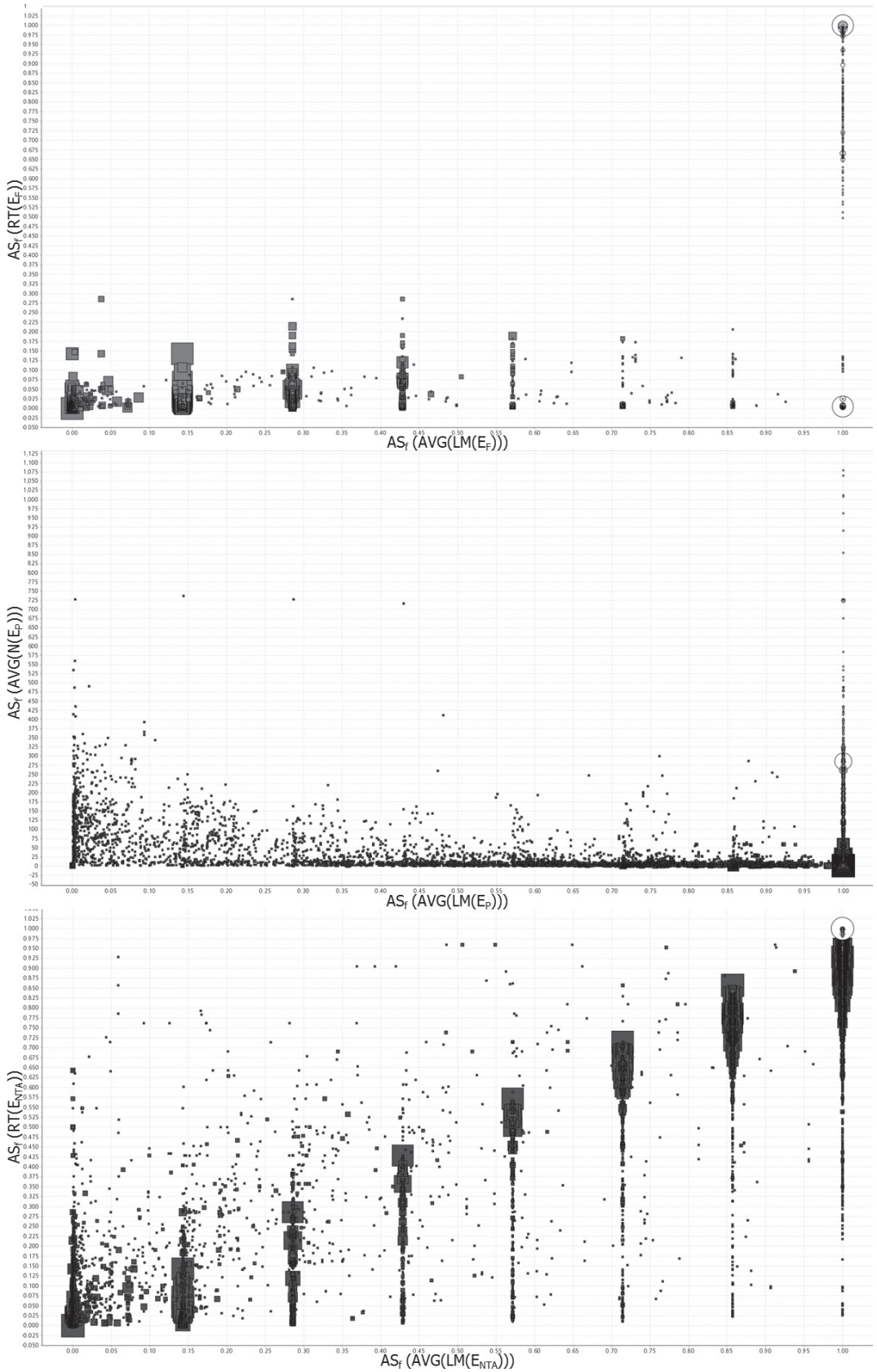


Fig. 11 Scatter matrix of raw data (gray rectangles for normal application and white circles for Internet worm).

rameter K is 3. Gray rectangles represent normal application and white circles do Internet worms respectively. The larger size rectangle and circle show more frequency of data. The features of $AS_f(AVG(LM(E_F)))$, $AS_f(AVG(LM(E_P)))$, $AS_f(AVG(LM(E_{NDA})))$ well explain the different behaviors of Internet worms from those of normal applications in feature space. In scatter matrix, small part of normal applications is located in the middle of Internet worms, and some of Internet worms and normal applications overlap each other. That is the reason why the false alarms occur. However, one-class SVM maps the aforementioned 6 features into a high dimensional feature space and successfully makes normal profile which shows dramatic good performance. In next chapter, we evaluate PC-WDS performance and how each of 6 features is significant to increase detection performance and reduce false alarms.

4.3 PC-WDS Performance and Significance of Each Detection Features

Table 4 shows the overall performance of PC-WDS. 6 features for detecting Internet worms and one-class SVM perfectly classify Internet worms (with 0 false negative alarm) and successfully reduce false positive alarms.

To evaluate each of 6 features, which are highly useful or which may be useless, we have another experiment. We assume that elimination of useless detection feature may enhance overall performance; however lack of significant feature may decrease overall detection performance. We apply the technique of deleting one feature at a time to evaluate 6 worm detection features and explain how the features are important ones for Internet worm detection using one-class SVM [29].

Table 5 shows the performance results of deleting one feature, and comparing Tables 4 and 5, it explains that:

- With 6 detection features PC-WDS detects all 23 Internet worms without false negative alarm, and makes 0.08% of false positive alarms.
- All of 6 detection features play an important role in detecting Internet worms and reducing false alarms.
- The most important features for detecting Internet worms are Ratio of New Destination Address and Likelihood of Machine-triggered for Failed Network Connection.
- The most important features for reducing false positive alarms are Likelihood of Machine-triggered for Port Creation, Ratio of New Destination Address and Ratio of Failed Network Connection.

Table 4 Overall performance of PC-WDS (with 6 detection features).

False positive ratio	Worm detection ratio	False negative ratio
0.08%	100%	0%

Table 5 Deletion of one feature at a time.

Feature deleted	False positive ratio	Worm detection ratio	False negative ratio
$AS_f(AVG(LM(E_F)))$	0.18%	99.80%	0.20%
$AS_f(RT(E_F))$	25.72%	100%	0%
$AS_f(AVG(LM(E_P)))$	32.62%	100%	0%
$AS_f(AVG(N(E_P)))$	0.17%	100%	0%
$AS_f(AVG(LM(E_{NDA})))$	3.23%	100%	0%
$AS_f(RT(E_{NDA}))$	29.88%	87.10%	12.90%

5. Conclusion

5.1 Possible Attacks on PC-WDS or Evasion Strategies

Worm developers are certainly technically savvy enough to understand the likely strategies used by worm detection software including PC-WDS, and they would surely do whatever they can to avoid or delay detection. We discuss likely scenarios and how design of PC-WDS deals with such attempts.

- Fake (e.g., software-generated) user events: As discussed earlier, monitoring of SendInput() API can tell if user events are "authentic", and current implementation of PC-WDS has such feature built-in. If a worm can run in the kernel mode, PC-WDS offers no protection. Additionally most of the recent operating systems provide access control mechanisms enough to access system resources based on object authentication policies such as SELinux [30] or sandbox, which are beyond our discussion in this paper.
- Fake destination addresses: Sneaky worms may use covert channels or proxy servers to perform scanning so that the same destination addresses appear repeatedly. Ratio of new destination addresses would then become low and the associated anomaly score lowered as well. To defeat such attempts, PC-WDS's data collection module analyzes SOCKS and Microsoft ISA proxy protocols to identify the real destination addresses. Unless a worm developer invents his or her own proprietary proxy protocol and communicates with a special proxy server using one's own encryption algorithm, fake destination addresses does not work. While theoretically imperfect, PC-WDS provides a strong enough deterrent.
- "Controlled attacks to maintain anomaly score below the threshold value": Worm developers may take detection algorithm of PC-WDS into consideration so that worms remain "patient enough" to limit scanning activities in such a manner that the anomaly score would never exceed the threshold value. PC-WDS will fail to detect such worms. However, worms would be forced to spread themselves at a much slower speed than desirable. Win32/Spybot.8pa!Worm, for example, made only 0.8 scanning attempts per minute. But, it may scan much too slowly to avoid detection by PC-WDS. Such worms do not pose serious threat. Furthermore, current design of PC-WDS demonstrated that it is effective enough to detect most of nasty and real worms.

5.2 Concluding Remarks

PC-WDS is a fully automated system to detect Internet worms. It does not require signatures, and empirical evaluation performed on real Internet worms clearly demonstrated that PC-WDS is effective. It extends relatively simple worm detection feature used in the BINDER system so that diversity of destination addresses and the patterns on port creations are analyzed in the context of the likelihood of them being initiated by user events. Users need not worry about configuration or parameter tuning, and performance overhead is practically negligible.

Contribution of PC-WDS is significant in that it provides adequate protection to users who do not possess technical knowledge on computers or security. While it is theoretically possible for worm authors to perform scanning, only a small number of network connections are allowed before it is detected by PC-WDS. Furthermore, it is a daunting technical challenge to develop worms that can successfully perform scanning activities while managing to avoid detection.

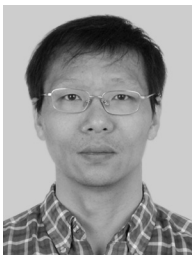
While current design of PC-WDS does not guarantee that it can detect all types of unknown worms in all circumstances, experiments results conducted using real-world worms were positive enough to convincingly demonstrate its capability. "Ordinary" users badly need adequate protection against worms, and PC-WDS provides adequate protection at almost no cost (e.g., CPU or memory requirements).

References

[1] D. Moore, C. Shannon, and J. Brown, "Code-Red: A case study on the spread and victims of an Internet worm," Proc. Second ACM SIGCOMM Workshop on Internet Measurement, pp.273–284, France, Nov. 2002.
[2] A. Mackie, J. Roculan, R. Russell, and M.V. Velzen, "Nimda worm analysis," Technical Report, Incident Analysis Report, SecurityFocus, Sept. 2001.
[3] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer worm," IEEE Security and Privacy, vol.1, no.4, pp.33–39, July 2003.
[4] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and J. Nazario, "The Blaster worm: Then and now," IEEE Security and Privacy, vol.3, no.4, pp.26–31, July/Aug. 2005.
[5] C. Shannon and D. Moore, "The spread of the Witty worm," IEEE Security and Privacy, vol.2, no.4, pp.46–50, July 2004.
[6] S. Staniford, V. Paxson, and N. Weaver, 'How to own the Internet in your spare time," Proc. 11th USENIX Security Symposium, San Francisco, USA, pp.149–167, Aug. 2002.
[7] Z. Chen, L. Gao, and K. Kwiat, "Modeling the spread of active worms," Proc. 22nd IEEE International Conference on Computer Communications, INFOCOM, pp.1890–1900, San Francisco, USA, March 2003.
[8] C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," Proc. 9th ACM Conference on Computer and Communication Security (CCS '02), pp.138–147, Washington, DC, USA, Nov. 2002.
[9] L. Spitzner, "The honeynet project: Trapping the hackers," IEEE Security and Privacy, vol.1, no.2, pp.15–23, March/April 2003.
[10] N. Provos, "A virtual honeypot framework," Proc. 13th USENIX Security Symposium, pp.1–14, San Diego, USA, Aug. 2004.
[11] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen, "Honeystat: Local worm detection using honeypots," Proc. 7th International Symposium on Recent Advances in Intrusion Detection (RAID '04), pp.39–58, France, Sept. 2004.
[12] C. Zou, W. Gong, D. Towsley, and L. Gao, "The monitoring and early detection of Internet worms," IEEE/ACM Trans. Netw., vol.13, no.5, pp.961–974, Oct. 2005.
[13] SANS, "SANS Internet storm center - Cooperative network security community," http://isc.sans.org, accessed May 12, 2013.
[14] J. Riordan, D. Zamboni, and Y. Duponchel, "Billy goat, an accurate worm-detection system (Revised Version) (RZ 3609)," Technical Report, IBM Zurich Research Laboratory, 2005.
[15] S. Singh, C. Estan, G. Varghese, and S. Savage, "The EarlyBird system for real-time detection of unknown worms," Technical Report, CS2003-0761, CSE Department, UCSD, Aug. 2003.
[16] J. Xia, S. Vangala, J. Wu, L. Gao, and K. Kwiat, "Effective worm detection for various scan techniques," J. Computer Security, vol.14, no.4, pp.359–387, July 2006.
[17] W. Cui, R.H. Katz, and W. Tan, "BINDER: An extrusion-based break-in detector for personal computers," Proc. USENIX Annual Technical Conference, pp.363–366, Anaheim, USA, April 2005.
[18] ICST of Peking University, "China honeynet project," http://www.icst.pku.edu.cn/honeynetweb/honeyneten/index.htm, accessed Sept. 10, 2012.
[19] Y. Kaplan, "API spying techniques for Windows 9x, NT and 2000," http://www.internals.com/articles/apispy/apispy.htm, accessed May 12, 2013.
[20] MSDN, "Microsoft developers network library," http://msdn.microsoft.com/library/, accessed May 12, 2013.
[21] C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining and Knowledge Discovery, vol.2, no.2, pp.121–167, 1998.
[22] N. Cristianini and J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, 2000.
[23] S. Mukkamala and A. Sung, "Feature ranking and selection for intrusion detection systems using support vector machines," Proc. 2002 Intl. Conf. on Information and Knowledge Engineering, pp.503–509, Las Vegas, USA, July 2002.
[24] W. Hu, Y. Liao, and V. Vemuri, "Robust support vector machines for anomaly detection in computer security," Proc. 2003 Intl. Conf. Machine Learning and Applications, pp.168–174, Los Angeles, USA, June 2003.
[25] D. Kim, H. Nguyen, and J. Park, "Generic algorithm to improve SVM based network intrusion detection system," Proc. 19th International Conference on Advanced Information Networking and Applications, no.2, pp.155–158, Taiwan, March 2005.
[26] B. Scholkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson, "Estimating the support of a high-dimensional distribution," J. Neural Computation, vol.13, no.7, pp.1443–1471, July 2001.
[27] C.C. Chang and C.J. Lin, "LIBSVM – A library for support vector machines," 2001, http://www.csie.ntu.edu.tw/˜cjlin/libsvm/, accessed May 12, 2013.
[28] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," Proc. 2003 ACM Workshop on Rapid Malcode (WORM '03), ACM SIGSAC, pp.11–18, Washington, DC, USA, Oct. 2003.
[29] A.H. Sung, "Ranking importance of input parameters of neural networks," Experts Systems with Applications, vol.15, no.3-4, pp.405–411, Oct./Nov. 1998.
[30] SELinux, "Security-enhanced Linux," http://selinux.sourceforge.net, http://fedoraproject.org/wiki/SELinux, accessed May 12, 2013.

Jeongseok Seo received an M.S. degree in the Computer Science at KAIST in 2002. His research interests include computer security and human interactions.

Sungdeok Cha is a professor in the Information and Communications at Korea University. He received a Ph.D. degree in the Information and Computer Science at the University of California, Irvine in 1991. From 1994 through 2007 he was a faculty in the Computer Science at KAIST. His research interests include software safety, computer security and formal method.



Bin Zhu is a lead researcher at Microsoft Research Asia. He received a Ph.D. degree in the Electrical Engineering at the University of Minnesota, Twin Cities. His research interests include digital rights management, P2P networks, web security and multimedia processing and communications.

Doohwan Bae is a professor in the Computer Science at KAIST. He received a Ph.D. degree in the Computer and Information Science at the University of Florida. His research interests include object-oriented technology, component-based software development, software process and autonomous systems.