FDF: Frequency detection-based filtering of scanning worms

Byungseung Kim\textsuperscript{a,}\textsuperscript{*}, Hyogon Kim\textsuperscript{b}, Saewoong Bahk\textsuperscript{a}

\textsuperscript{a} School of Electrical Engineering and Computer Science, INMC, Seoul National University, Seoul, Republic of Korea
\textsuperscript{b} Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea

\begin{abstract}
In this paper, we propose a simple algorithm for detecting scanning worms with high detection rate and low false positive rate. The novelty of our algorithm is inspecting the frequency characteristic of scanning worms instead of counting the number of suspicious connections or packets from a monitored network. Its low complexity allows it to be used on any network-based intrusion detection system as a real-time detection module for high-speed networks.

Our algorithm need not be adjusted to network status because its parameters depend on application manners have been a critical issue for mitigating its malignant impacts.

Although many defense techniques against scanning worms have been developed, they have difficulty in determining the threshold over which the suspicious behavioral patterns described above are positively identified. In fact, most detection algorithms need to tune their parameters to fit the environment they work in, such as the site and time-of-day characteristics for efficiency and accuracy. To make matters worse, legitimate Internet services behaving like worms and dynamic network environments undermines the efficacy of the techniques.

In this paper, we propose a scanning worm detection algorithm, named Frequency Detection-based Filtering (FDF). It achieves high detection rate and low false positive rate even when the scanning traffic from worms are mingled with legitimate scanning-type flows such as P2P traffic. Moreover, it can be easily implemented on the top of any existing network-based intrusion detection system (IDS) thanks to its simplicity.

The main idea of the FDF comes from the observation that scanning worms pause for a specific and characteristic period of time between individual scan attempts. TCP-based scanning worms usually transmit SYN packets at the rate prescribed by its self-propagation code. In contrast, normal TCP-based applications send SYN packets to other hosts at a rather indeterminate rate. This creates the different frequency characteristics that can be leveraged to distinguish the one from the other. The FDF extracts this frequency characteristics from just SYN arrival patterns from the monitored network, irrespective of the number of SYN packets.

The rest of the paper is organized as follows. We first observe the frequency characteristics of scanning worms through autocorrelation and power spectral density (PSD) estimates in Section 3. Then, we propose our FDF algorithm that detects the frequency characteristics in a simple manner and minimizes false positives. Section 5 compares the performances of our algorithm and SNORT. Then, we discuss some limitations and extension issues of our algorithm in Section 6, followed by the conclusion in Section 7.
\end{abstract}

1. Introduction

Recently, worm epidemics have become a grave concern by demonstrating their formidable power to incapacitate various Internet services and exhaust network resources. For instance, CodeRed, Blaster, Nimda, and SQL Slammer worms inflicted huge economic and social damages, and their mutations are still threatening the Internet environment. A distinct feature of the worms is their self-propagation behavior that is enabled by fast, automated scanning for possible victims. They can even spread globally in just a few minutes [1,8,11]. Furthermore, the technique automatized scanning for possible victims. They can even spread worms is their self-propagation behavior that is enabled by fast, threatening the Internet environment. A distinct feature of the

1. Introduction

Recently, worm epidemics have become a grave concern by demonstrating their formidable power to incapacitate various internet services and exhaust network resources. For instance, CodeRed, Blaster, Nimda, and SQL Slammer worms inflicted huge economic and social damages, and their mutations are still threatening the Internet environment. A distinct feature of the worms is their self-propagation behavior that is enabled by fast, automated scanning for possible victims. They can even spread globally in just a few minutes [1,8,11]. Furthermore, the technique automatized scanning for possible victims. They can even spread worms is their self-propagation behavior that is enabled by fast, threatening the Internet environment. A distinct feature of the

1. Introduction

Recently, worm epidemics have become a grave concern by demonstrating their formidable power to incapacitate various internet services and exhaust network resources. For instance, CodeRed, Blaster, Nimda, and SQL Slammer worms inflicted huge economic and social damages, and their mutations are still threatening the Internet environment. A distinct feature of the worms is their self-propagation behavior that is enabled by fast, automated scanning for possible victims. They can even spread globally in just a few minutes [1,8,11]. Furthermore, the technique automatized scanning for possible victims. They can even spread worms is their self-propagation behavior that is enabled by fast, threatening the Internet environment. A distinct feature of the

1. Introduction

Recently, worm epidemics have become a grave concern by demonstrating their formidable power to incapacitate various internet services and exhaust network resources. For instance, CodeRed, Blaster, Nimda, and SQL Slammer worms inflicted huge economic and social damages, and their mutations are still threatening the Internet environment. A distinct feature of the worms is their self-propagation behavior that is enabled by fast, automated scanning for possible victims. They can even spread globally in just a few minutes [1,8,11]. Furthermore, the technique automatized scanning for possible victims. They can even spread worms is their self-propagation behavior that is enabled by fast, threatening the Internet environment. A distinct feature of the
known to be vulnerable to buffer-overflow. In this process, they cause high rate of failed connections. These can be verified by gathering ICMP unreachable messages in the monitored network [10] or analyzing highly anomalous traffic relative to the usual traffic distribution [5,6,17].

This characteristic of scanning worms is the most common rule deployed to existing IDses to detect scanning worms. Specifically, in SNORT which is a well-known open-source IDS, there are two portscan modules: portscan and afportscan preprocessors modules that are designed for detecting and tracking portscans [14,15]. The "portscan" module counts new connection requests from each host for a given period. If the counter for a host exceeds the threshold, it regards the host as a scanner or worm. The afportscan detects a port scan by tracking connections and negative responses that are kinds of error packets such as RST and ICMP packets in querying connections from the target host to other hosts. This module can be very effective to detect port scan because negative responses are hardly generated by legitimate hosts.

However, there can be complications in detecting scanning worms. First, it is not easy to determine the threshold over which the suspicious behavioral patterns described above are positively identified. In fact, most detection algorithms need to tune their parameters to fit the environment they work in, such as the site and time-of-day characteristics for efficiency and accuracy. Second, some internet services that show similar behaviors with the worms are likely to cause false positives. Especially, some lower rate of scan compared to that of connection requests from recent legitimate services baffle rate-based detection algorithms, which are used by many intrusion detection systems (IDses), to detect scanning worms with low threshold setting, resulting in many false positives. For instance, a P2P client often behaves like a scanning worm when searching for P2P servers that have desired contents. Moreover, negative responses cannot be expected to be utilized since Windows XP servicepack2 drops all SYN packets to its closed ports silently, and most intermediate routers enhanced by security policies do not send ICMP-T3 messages but silently drop SYN packets as responses to connection requests to unallocated IP addresses.

There have been several work not considering the above rate-based characteristic of scanning worms. In [2,3], since scanning worms frequently use uniformly distributed IP addresses as their target hosts in random IP scanning for finding vulnerable hosts, they expose some specific packet flows between the monitored network and the Internet. Due to these idiosyncrasies, scanning worms are likely to cause false positives. Especially, some lower rate of scanning can be observed. In this case, the scanning interval does not have to be an exact multiple of the sampling interval since if the scanning interval belongs to the sampling interval for counting the number of SYN arrivals from legitimate TCP sessions and a scanning worm in the interval \( t_{n-1} - t_n \), respectively, where \( t_{n-1} = t_n - T \) is the sampling interval for counting the number of SYNs. Then,

\[
P(X_n = k) = \frac{(\lambda T)^k e^{-\lambda T}}{k!}, \quad \text{for } k = 0, 1, 2, \ldots.
\]

where \( N_w \) is the number of active threads created by an infected host, and \( N_w \) is the scanning period of the worm. In real estimate, the scanning interval does not have to be an exact multiple of the sampling interval since if the scanning interval belongs to the nth sampling interval, the scanning is counted as \( Y_n = A_w \). Also, \( N_w \) should be greater than one to obtain the frequency characteristic of this model. For instance, if the scanning interval spans two sampling intervals, its frequency cannot be observed. This can be resolved by letting the sampling interval \( T \) be smaller than the half of the scanning period, i.e., \( T_w = N_w - T \). This will be discussed further in the next subsection.

The summation of \( X_n \) and \( Y_n \) implies that the traffic containing scanning worms is observed at the gateway. That is, we should find a periodical characteristic \( X_n \) buried under a legitimate traffic \( Y_n \) in order to detect scanning worms. Therefore, we use autocorrelation estimate which is a mathematical tool for finding repeating patterns [21]. For a discrete process \( \{X_n, X_{n-1}, \ldots, X_k\} \) with a known mean \( \mu \) and variance \( \sigma^2 \), the autocorrelation estimate in terms of a sample lag \( m \) is obtained as

\[
R(m) = \frac{1}{N - |m|} \sum_{t=m+1}^{N} [X_t - \mu][X_{t-m} - \mu]. \tag{1}
\]

Following Eq. (1), the autocorrelations of \( X_n \) and \( Y_n \) are given by

\[
R_{\text{X}}(m) = (iT)^2 + iT\delta(m),
\]

\[
R_{\text{Y}}(m) = \frac{1}{N - |m|} \sum_{n=0}^{N-m} Y_{n+m} Y_n = \frac{A_w^2}{N_w} \delta(m - kN_w).
\]

Letting \( Z_n = X_n + Y_n \), we obtain

\[
R_{\text{Z}}(m) = (iT)^2 + 2iT\frac{A_w}{N_w} + iT \cdot \delta(m) + \sum_{k=0}^{N_w-1} \frac{A_w^2}{N_w} \delta(m - kN_w). \tag{2}
\]

The first two terms of \( R_{\text{Z}} \) are independent of the sample lag and have some constant value on the autocorrelation estimate, and the third term is zero for \( m = 0 \). The last term of \( R_{\text{Z}} \) is related with the worm and shows the impulse train. Because of this, we can differentiate a worm scanning from legitimate SYN arrivals. Furthermore, inspecting the impulse train of the autocorrelation, we can find out the corresponding frequency of the scanning. To get the precise frequency characteristic, we can apply PSD estimate which the Fourier transform of the autocorrelation function, \( R(m) \) [21].

We applied the autocorrelation estimate to real traces. The traces used in our experiments have been collected by TCPDUMP at the gateway router between a university and the Internet for...
25 h.² A summary of our traces is given in Table 1. The ‘Outgoing’ in the table represents unidirectional packet departures from the university to the Internet. ‘03_out’ denotes the outgoing of our trace for the year 2003, and ‘04_in and ‘04_out denote the incoming and outgoing of our trace for the year 2004, respectively.

To check whether any known worms exist in our traces, we applied the signatures of SNORT2.0 before experiments, and we identified two hosts with CodeRedII in the 2003 trace but none in the 2004 trace. In fact, there were several hosts suspected of some type of infection in the 2004 trace. But we could not identify the worm(s) by signature matching because there were no successful setup of TCP connections over which to transport exploit codes.

In the measurement of autocorrelation and PSD, each sample represents the number of SYN arrivals for a time bin of 10 ms, and one sample set consists of 10,000 samples. We obtained 912 sample sets in the 2003 trace alone and 880 × 2 sample sets in the 2004 trace. Fig. 2 shows the example results for the 15th set in the 2003 trace. To investigate the effect of CodeRedII in the 2003 trace, we perform the autocorrelation and psd estimates before and after excluding SYN arrivals of the two infected hosts from the trace. The results are shown in Fig. 2. Clearly, the autocorrelation provides a more visible indication on the existence of scanning worms than the time series of a sample set. In Fig. 3, we show a PSD estimate of the time series. We notice that there are SYN arrivals from the worm with the frequency of 1/Hz that bears out the periodicity of 10 ms intervals. In our experiments, we set the sampling period T = 10 ms that enables us to detect the worm frequency of up to 50 Hz.

Third, the state of the monitored network should be stable. The fixed propagation delay between an infected host and the monitoring system does not affect the autocorrelation estimate, but the variable latency caused by network congestion influences the frequency characteristic very much. Since the network seems to operate normally at least for a couple of minutes even in a high-speed network while worms propagate throughout the network, it is a feasible assumption that the network latency keeps up with a steady state, which allows us enough time to measure the frequency characteristic.

Lastly, we have to consider the processing time of the autocorrelation and PSD estimates for real-time detection. Since the observable frequency is proportional to the sample lag m and the sampling interval T, the autocorrelation with m lags needs to be computed m × n times where n is the number of samples. To obtain the PSD estimate, we need to perform the Discrete Fourier Transform (DFT) for the obtained m lags of the autocorrelation. Accordingly, the autocorrelation estimate runs in O(mn) and the PSD estimate in O(m log m). In our experiments, we set m = 100, that is, 100 ms, and T at 0.01. Then, we can detect the worm having scanning frequency of 1 = 100 Hz through 50 = 1/0.01 × 10 ms intervals. From these parameters, the autocorrelation estimate and the PSD estimate are just computed 100 × 10,000 times and 100 × 10,000 + 100 log10 times, respectively.

3 CodeRedII probes IP addresses randomly for a eighth of the time. It probes machines in the same /8 network and the same /16 network for a half of the time and three eighth of the time, respectively [18].

4. Frequency detection-based filtering

To distinguish infected hosts from legitimate hosts, we perform the autocorrelation estimate of SYN arrivals on individual host basis. The complexity of the autocorrelation estimate with m lags and n samples is O(mn), but with N hosts to inspect it is O(Nmn). To cope with the high complexity, here we propose a O(1) method for the FDF.

In case the monitor is at the periphery of the monitored network, which is usually the case, local scanning is not visible to the monitor and affects the accuracy of the autocorrelation estimate. But a worm should perform some level of global scanning unless it wants to remain in the infected network. Letting Tw be the scanning period of a thread, we can obtain the probability of the SYN arrivals at the monitoring point as

\[ P_{\text{SYN}} = 1 - \left(1 - P_a + P_s \frac{M_w}{2T_w}\right)^{A_w} \]
Fig. 1. Time series of a sample set in the 2003 trace.

Fig. 2. Autocorrelation of Fig. 1 for the sample lag of \(-100\) through \(+100\) in the unit of 0.01 s.

Fig. 3. PSD estimate of Fig. 1.
4.1. FDF algorithm

Algorithm 1 shows the pseudo code of the FDF that has the following features:

- Time slot \( T_s \) — This is determined according to the autocorrelation estimate. For example, if the autocorrelation estimate is to detect the SYN scanning at 10 Hz, we set the time slot length to 0.1 s.
- Hash table entry — A distinctive characteristic of scanning worms is that they scan a specific port known to be vulnerable. That is, SYN packets from an infected host have one source IP (SIP), many destination IPs (DIPs), many source Ports (SPs) and a single destination Port (DP). Therefore, we define a “flow” as SYN arrivals with the same SIP and DP pair. The FDF creates a hash table with each entry key consisting of the SIP and DP, and updates it according to the arrival of a new DIP. An entry of the hash table consists of a duration counter \( n_d \), which counts the number of consecutive time slots of observing a flow, the time stamp of starting time slot, \( t_s \), for the flow, the time stamp of latest updated time slot, \( t_l \), and SYN counter \( n_s \), indicating the number of SYN arrivals during the current time slot.
- Duration threshold \( D_{th} \) — If the worm scanning lasts more than \( D_{th} \), the FDF raises an alarm.
- Update margin \( U_m \) — FDF keeps counting even if there is no scanning activity for less than \( U_m \) time slots. This handles the heavily local scanning worms with minimal global scanning component. Unless they are properly handled, they could cause false negatives.

Algorithm 1 Frequency detection-based filtering

1: Procedure FDF(each SYN packet) 2: Initialize 3: for SYN arrivals do 4: if Hash(SIP,DP) then 5: if \( t - t_s < U_m \) then 6: Update_table_entry(SIP,DP) 7: \( n_s \leftarrow n_s + 1 \) 8: \( t_s \leftarrow t \) 9: else 10: Initialize_table_entry(SIP,DP) 11: \( t_s \leftarrow t_s + t \) 12: \( n_s \leftarrow 1 \) 13: end if 14: else 15: Initialize_table_entry(SIP,DP) 16: \( t_s \leftarrow t_s + t \) 17: \( n_s \leftarrow 1 \) 18: end if 19: if \( n_s > D_{th} \) then 20: Alarm(SIP,DP) 21: end if 22: end for 23: end procedure

In Algorithm 1, the number of consecutively observed time slots is sequentially labeled by an integer number, \( t \), which implies the time event stamp from the starting of the algorithm. Namely, \( t_s \) and \( t_l \) do not denote the real packet arrival time, but the slot numbers for the \( t \)th event for the SYN packet interested. From line 3 to line 18, the FDF updates \( n_s \), \( t_s \), and \( t_l \) according to the present time slot number. If the source’s IP and its port number are newly observed, or the number of slots counted from the latest updated time slot is more than the update margin \( (U_m) \), the FDF initializes a table entry for the observed source IP. Then, from line 19 to line 21, if \( n_s > D_{th} \), the FDF reports a scanning worm detected, otherwise continues to observe it.

The FDF detection system has three stages: pre-detection, detection, and defense. In the pre-detection stage, the system determines whether the frequency characteristic of a scanning worm appears through the autocorrelation estimate. If the frequency characteristic is detected, the FDF locates infected hosts in the detection stage. Finally, in the defense stage, the information about the infected hosts is handed over to the defense system and the infected hosts are isolated from the network.

4.2. Behavior of SYN arrivals from legitimate hosts

For optimal operation of the FDF system, we need to obtain the system parameters that allow us to tell the SYN arrivals caused by scanning worms from those by legitimate hosts. To do so, here we investigate the properties of the SYN arrivals from web clients and P2P clients that are the two dominating applications today.

These days P2P traffic accounts for a considerable share of the Internet traffic. Interestingly, hybrid P2P systems perform SYN scanning like a worm. In such systems the P2P client sends SYN packets to the servers on the list obtained from the P2P master server to check their liveness and round-trip time (RTT) before searching files or refreshing the server list. So, it may well cause false positives. With our Internet traces, we attempted estimating the SYN interarrival time distribution for a P2P client when it searches P2P servers. Our traces contain many popular P2P traffic traces produced by such applications as e-donkey, Soribada, KaZaa, V-share, and File-Guri (some are local versions). Fig. 5(a) shows the CDF of SYN interarrival time of a V-share client fitted against the exponential distribution with \( \lambda = 37 \) (packets/s). Fig. 5(b) is the QQ-plot which indicates the degree of similarity between two distributions. Almost linear line in the QQ-plot verifies that the SYN interarrival time of V-share is approximately exponentially distributed. Other P2P clients also exhibit similar characteristics with various mean rates of 10 through 60 (packets/s) during each SYN burst.

We also observe from our traces that when a web client issues multiple http requests for embedded objects on a web page, it generates about 5–80 outgoing SYN packets in a burst. This type of burst SYN arrivals can cause false positives in the FDF system. In this paper, however, we do not address the problem because P2P clients cause much more false positives than web clients and because there are many prior work on the subject [12,13].

Now we investigate the burst duration and the SYN arrival rate of P2P and web clients in a SYN burst. We represent the burst event of a flow as SYN packets that arrive at the rate of more than 10 (packets/s) when it is active and it have the idle period of less than 1 (s) since the human interaction causing the idle period commonly takes more than 1 (s). The summary of our investigation is given in Table 2. Fig. 6 shows that the SYN arrival rate in a burst is smaller than 60 (packets/s), and the burst most likely lasts less than 8 (s).

4.3. Parameter adjustment

According to the observation of legitimate SYN arrivals described above, we adjust the parameters of the FDF: time slot length, duration threshold, and update margin.

4.3.1. Time slot length

The time slot, \( T_s \), is determined by the autocorrelation estimate or the PSD estimate. As shown in Fig. 2(a), the autocorrelation estimate of legitimate SYN arrivals exhibits some fluctuation. How-
(a) The measured CDF of SYN interarrival times from a P2P client (V-share) fitted by the exponential distribution with $\lambda = 37$ (packets/sec)

(b) Q-Q plot

Fig. 5. Distribution of SYN interarrival time of P2P traffic.
ever, we can clearly differentiate the impulse train of a scanning worm from the reference value of legitimate traffic. Let us take the reference value to be the minimum of $R_{zz}^{m}$, and denote it by $R_{zz}^{\min}$. Considering error and noise denoted by $e(m)$, we can represent $H(m)$ as a likelihood function for decision as

$$H(m) = \frac{R_{zz}^{m} - R_{zz}^{\min}}{R_{zz}^{\min}} = \begin{cases} \frac{\frac{1}{\lambda} - \frac{e(m)}{\lambda}}{\frac{1}{\lambda} - \frac{e(m)}{\lambda}} & \text{if } m = m^{'}, \\ \frac{\frac{1}{\lambda} - \frac{e(m)}{\lambda}}{\frac{1}{\lambda} - \frac{e(m)}{\lambda}} & \text{if } m\neq m^{'}, \end{cases}$$

where $m^{' }$ is the sample lag at which the impulse train appears on the autocorrelation estimate.

In this equation, we have to find $m^{'}$ satisfying $H(m^{'}) > h$ with small error where $h$ represents a fluctuation threshold. $h$ is large enough to ignore the impact of $e(m)$ since $h$ is the total arrival rate of legitimate SYN arrivals. Accordingly, $H(m)$ for $m = m^{'}$ is kept small in comparison with $H(m)$. In our experiments, the maximum value of $H(m)$ for $m = m^{'}$ was about 0.2. Thus, we set a fluctuation threshold, $h$, at 0.3 by adding some safety margin to reduce the error possibility. Our measured value of $h$ can be applied to other networks because it does not depend on the monitored network but the application services. Considering the variation in $h$, we group all the $m^{'}$s according to the range of autocorrelation value. Intuitively, if we have a group of $(m_{1}, m_{2}, m_{3}, \ldots)$ that have similar autocorrelation values within the variation $h$, we obtain the frequency characteristic that has the period $\tau$ and satisfies $m_{k}^{'} \in \{m|m = 2k + m_{0}; k = 0, 1, 2, 3, \ldots\}$. If there are too many groups, we can use the PSD estimate as a tool for investigating the frequency characteristics. The FDF sets the time slot length to the maximum among these obtained periods, $\tau$. This is because SYN arrivals that have some other short interarrival times can be continuously observed within the maximum period.

### 4.3.2. Duration threshold

The duration threshold, $D_{th}$, is the minimum period for continuous monitoring. This is needed to differentiate SYN arrivals generated by a worm from those by legitimate traffic. If the threshold is too low, the FDF would brand many legitimate hosts as infected. Namely, if there are many P2P clients that search P2P servers, the FDF would generate many false positives. In contrast, if we set it too high, there exists a possibility of false negatives. Therefore, we need to find the optimal threshold. To do so, we take advantage of the features of legitimate SYN arrivals analyzed in Section 4.2. Since it is likely that P2P clients cause more false positives than web clients, we focus on P2P clients.

Since SYN arrivals from a P2P client can be approximated by the Poisson distribution, we can obtain the probability that the number of observed SYN arrivals, $k$, from the P2P client for $T_{s}$ is greater than $n$ as

$$p_{a} = p[k > n] = 1 - \sum_{k=0}^{n} \frac{e^{-\lambda T_{s}(\lambda T_{s})^{k}}}{k!},$$

where $\lambda$ is the average SYN arrival rate in a burst. In our experiments, we set $\lambda$ to 60 packets/s that gives the CCDF probability of less than 0.01 in Fig. 6(a).

Since P2P and web clients keep generating SYN packets for a relatively short duration as shown in Fig. 6(b), they do not have to be monitored for $max_{duration}$. In contrast, a worm performs scanning for a somewhat long time, so it needs to be monitored for a $min_{duration}$ at least. As the FDF operates in the time slotted manner, we can denote the $max_{duration}$ and the $min_{duration}$ by $d_{max}$ and $d_{min}$ in the unit of a time slot, respectively. Accordingly, for a given target probability $P_{f}$ of false positives, we can obtain an optimum duration threshold $D_{th}$ as the following.

$$D_{th} = min\{d|p_{a}(n)^{d} \leq P_{f}\} \quad \text{for } d_{min} \leq d \leq d_{max},$$

where $n$ is the number of observed SYN arrivals in a time slot. This equation tunes $D_{th}$ according to the quantity of observed SYN arrivals for the smart detection. For example, if a worm sends SYN packets faster than a legitimate host does, a short $D_{th}$ makes FDF detect it more quickly. On the contrary, a long $D_{th}$ provides against false positives in detecting a slow-scanning worm. In our experiments, we set the $max_{duration}$ and the $min_{duration}$ to 8 s and to 0.5 s, respectively, which has the $P_{f}$ of 0.01 in Fig. 6(b). That is, if $T_{s}$ is 0.1, $d_{max}$ is 5 ($=0.5/0.1$) and $d_{min}$ is 80 ($=8/0.1$). We can reduce false positives of the FDF by increasing $d_{max}$. However, the larger $d_{max}$ is, the larger the...
5.1. FDF against real-life traces

From the analysis of our traces with the parameter adjustment in Section 4, we obtained the FDF parameters as shown in Table 3 and used them for evaluating the FDF. We used three traces including an incoming traffic. For the incoming trace, the FDF cannot observe enough SYN arrivals from the hosts it is interested in, so we evaluated the performance of FDF with respect to two different parameter sets. For ‘04_in(a), we use the same set as for the outgoing trace, and for ‘04_in(b), we use the set with the lower values of \( T_s \) and \( d_{\text{min}} \) to increase the sensitivity of FDF.

To investigate how many scanning worms are in the traces, we applied SNORT2.0 that executes the worm signature detection and the spp_portscan detection module. Then, we manually compared the detected flows with those obtained by the signature behaviors given in the online worm libraries [9]. As a result, we found two hosts infected by CodeRedII from ‘03_out and six infected hosts from ‘04_out. To identify any false positives, we examined the packet payload and DIPs for each detected flow. We found several false positives that are created by http, SMTP, and P2P. In the case of the detected SMTP flow of ‘04_out, we noticed that there are many HTML contents like spam mails and all the DIPs correspond to legitimate SMTP servers. In the same manner, it was easy to verify the remaining false positives by inspecting the port numbers and the delivered contents.

Table 4 summarizes the results of our experiments. Except for the CodeRedII, most of the detected scanning flows exhibited sequential scanning behavior. As they failed to establish connections, we could not determine what kind of worms they were. Assuming that only a worm performs SYN scanning to infect other hosts, we inferred the type of each worm by investigating the target port list and showed it on the second column [9]. The number of detected flows is not the same as that of detected hosts because recent worms such as Sasser and Agobot construct multiple flows to scan multiple service ports. The FDF detected all the scanning worms in ‘03_out, ‘04_out, ‘04_in(b), but not in ‘04_in(a). Considering that only a small portion of scanning SYN arrivals can be ob-

### Table 3

A summary of obtained FDF parameters for real traces.

<table>
<thead>
<tr>
<th>Parameter set</th>
<th>( T_s )</th>
<th>( \varphi )</th>
<th>( d_{\text{max}} )</th>
<th>( U_m )</th>
<th>( P_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘03_out</td>
<td>0.1</td>
<td>60</td>
<td>20</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>‘04_out</td>
<td>0.6</td>
<td>60</td>
<td>80</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>‘04_in(a)</td>
<td>0.1</td>
<td>60</td>
<td>30</td>
<td>480</td>
<td>6</td>
</tr>
<tr>
<td>‘04_in(b)</td>
<td>0.1</td>
<td>20</td>
<td>10</td>
<td>480</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 4

A summary of the results of FDF for real traces.

<table>
<thead>
<tr>
<th>Trace</th>
<th>Detected port</th>
<th>Flows</th>
<th>Rate (#/s)</th>
<th>Duration (s)</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘03_out</td>
<td>80 (CodeRedII)</td>
<td>2</td>
<td>73.7</td>
<td>6072.4</td>
<td>Random Port Scan</td>
</tr>
<tr>
<td></td>
<td>8404 (v-share)</td>
<td>3</td>
<td>45.6</td>
<td>192.0</td>
<td>False Positive(P2P Scan)</td>
</tr>
<tr>
<td>‘04_out</td>
<td>1025 (W32.Keco)</td>
<td>1</td>
<td>35.0</td>
<td>11664.0</td>
<td>/24 Sequential Scan(1KB payload)</td>
</tr>
<tr>
<td></td>
<td>3140 (Optix)</td>
<td>5</td>
<td>19.7</td>
<td>5275.8</td>
<td>/24 Sequential Scan(1KB payload)</td>
</tr>
<tr>
<td></td>
<td>80 (http)</td>
<td>1</td>
<td>3.05</td>
<td>96.0</td>
<td>False Positive</td>
</tr>
<tr>
<td></td>
<td>25 (SMTP)</td>
<td>1</td>
<td>2.3</td>
<td>144.0</td>
<td>Spam Mail</td>
</tr>
<tr>
<td>‘04_in(a)</td>
<td>9898 (W32.Daver)</td>
<td>2</td>
<td>235</td>
<td>0.6</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>4899 (W32.Agobot)</td>
<td>1</td>
<td>160</td>
<td>21.0</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>135 (W43.Blast)</td>
<td>3</td>
<td>40</td>
<td>96.0</td>
<td>/16 Sequential Scan(no connection)</td>
</tr>
<tr>
<td>‘04_in(b)</td>
<td>6129 (W32.Mockbot)</td>
<td>2</td>
<td>80</td>
<td>6.0</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>1433 (SQL worm)</td>
<td>1</td>
<td>171.8</td>
<td>82.4</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>9898 (W32.Daver)</td>
<td>9</td>
<td>221.6</td>
<td>0.5</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>1023 (W32.Sasser)</td>
<td>2</td>
<td>250.0</td>
<td>0.3</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>5554 (W32.Sasser)</td>
<td>4</td>
<td>220.0</td>
<td>0.3</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>135 (W32.Blast)</td>
<td>3</td>
<td>34.8</td>
<td>132.0</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>4899 (W32.Agobot)</td>
<td>2</td>
<td>314.8</td>
<td>22.1</td>
<td>/24 Sequential Scan(no connection)</td>
</tr>
<tr>
<td></td>
<td>9493 (File-Guri)</td>
<td>1</td>
<td>18.6</td>
<td>0.7</td>
<td>False Positive(P2P Scan)</td>
</tr>
</tbody>
</table>
served from the incoming traffic, the FDF needs to use a small duration threshold that can be controlled by a lower \( k_b \) like that of '04_in(b). However, a small duration threshold can cause many false positives. In our experiments, no false positives are found in both '04_in(a) and '04_in(b) traces. That is, the probability of false positives on the incoming trace is much smaller than that on the outgoing trace. This is because most Internet servers are located outside the monitored AS. In other words, the FDF can detect scanning flows from the incoming traffic without false positives, but needs careful setting of parameters and discrete investigation of SYN arrivals from the both directions.

![Visualization of Port Scanning Flows detected by FDF.](image)

![Performance comparison of false positives and detected scanning worms in '04_out trace.](image)
the interval of the spp_portscan to duration of longer than 60 s. Considering its detection time, we set parameters, the result of '04_out trace shows a same optimal value for a positives is minimal without having false negatives. In our experi-
ments, the optimal values make some points representing short durations. In Fig. 8(c) of the incoming trace, the detection is based on a small portion of SYN arrivals from an infected host, so all the detected flows have short durations.

5.2. Comparison with SNORT

5.2.1. False positives
To compare the performance of the FDF with that of SNORT, we applied the spp_portscan module of SNORT2.0 to '04_out trace. The spp_portscan module counts new connection requests from a host for a specific interval. If the count is greater than the threshold, the module regards the host as a scanner or a worm. For instance, we set the two parameters of the threshold and the interval to 90 and 30, which are denoted by 90/30, the module detects the host that attempts to make connections more than 90 times within 30 s. Be-
cause there is no reference concerning the parameter set of spp_portscan, we varied the parameters from extremely low to high and found the optimal values at which the number of false positives is minimal without having false negatives. In our experi-
ments, the result of '04_out trace shows a same optimal value for a duration of longer than 60 s. Considering its detection time, we set the interval of the spp_portscan to $x/60$ where $x$ is the threshold value of the number of SYN packets observed in 60 s.

In Fig. 9, we represent the relation between the numbers of false positives and detected scanning worms. In the case of the FDF, the number of detected scanning worms is fixed at six even if we in-
crease the max_duration. In the case of SNORT, both the numbers of false positives and detected scanning worms decrease as the threshold increases. Consequently, the spp_portscan generates more false negatives with the increase of the threshold while false positives are reduced. In essence, the FDF is more effective in detecting scanning worms than the spp_portscan of SNORT with less false positives.

5.2.2. Detection speed

Fig. 10 visualizes the characteristics of scanning worms com-
pared with those of legitimate traffic. In Fig. 8(a) and (b), most of the scanning worms create long lines, and most of the false posi-
tives make some points representing short durations.

Fig. 8 visualizes the characteristics of scanning worms com-
pared with those of legitimate traffic. In Fig. 8(a) and (b), most of the scanning worms create long lines, and most of the false posi-
tives make some points representing short durations. In Fig. 10, we represent the relation between the numbers of false positives and detected scanning worms. In the case of the FDF, the number of detected scanning worms is fixed at six even if we in-
crease the max_duration. In the case of SNORT, both the numbers of false positives and detected scanning worms decrease as the threshold increases. Consequently, the spp_portscan generates more false negatives with the increase of the threshold while false positives are reduced. In essence, the FDF is more effective in detecting scanning worms than the spp_portscan of SNORT with less false positives.

5.2.2. Detection speed

Fig. 10 shows the detection times of the FDF and SNORT with the optimal parameters that give six detected worms and four false positives. The FDF is slower than SNORT for the low scan rate of worms. However, it guarantees the upper bound of the detection time as the max_duration and detects a high-rate scanning worm as fast as SNORT. The upper bound satisfies the general requirements for containing low-rate scanning worms [1]. In the case of $U_m = 1$, the FDF detects a low-rate scanning worm faster than SNORT as shown in Fig. 10(b). That is, if the FDF is deployed to detect global scanning worms in a monitored network of small size which does not require the update margin, it can achieve its best performance.

6. Discussions

6.1. Extension

The FDF could be used for detecting periodic attacks other than scanning worms. For example, since most SYN flooding attacks are automated by scripts or worms, they expose some frequency characteristics that can be detected by the FDF. Such an example is the Blaster worm that is programmed to execute the SYN flooding at-
tack sending SYN packets to a target host every 20 ms [20], and then each infected host sends SYN packets to another victim in the same manner. Therefore, even if there seems to be no suspi-
cious behavior in the monitored network, DDoS attack can be launched by many infected hosts. In this case, the FDF can detect the SYN flooding attack and start to drop SYN packets from the attackers before the DDoS attack reaches a target host.

Also, the FDF can be applied for detecting scanning worms using UDP protocol such as SQL Slammer [7]. To verify this, we tried to execute the FDF on UDP packets. Since there does not exist this type of attack in our real-life traces, we injected virtual UDP scan-
ning trace with a deterministic sleep time into the '04_out trace. The FDF detected all the virtual UDP-based worms, but there were many false positives. This is because most P2P and VOD servers de-
lever data periodically through the UDP protocol. These false alarms can be removed by introducing additional rules. For exam-
ple, all the packets of a scanning flow have the same payload that is an exploit code of a worm, but those of a legitimate flow do not. So, we can compare the payload checksum to see whether the packet has the same (malicious) payload. Consequently, the FDF could be a viable solution to unknown UDP-based worms.

6.2. Limitation

We now discuss about what happens if an attacker programs a worm self-propagation code to have a non-deterministic sleep
SYN with which each decides sending a SYN with probability \( p \). A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution. A scanning worm generating SYNs with a random time interval can be implemented by showing some probability distribution.

The probability curve that more than one SYN is observed at a looping round according to the intended scanning rate. The authors introduce an algorithm, FDF (Fast Deterministic Scanning worm Detection), for detecting scanning worms. The FDF algorithm exploits the frequency characteristic of scanning worms and their mutations emitting the frequency characteristic. Notice that the frequency characteristic is independent of the number of SYN packets. So, the FDF can detect slow scanning worms as well as fast scanning worms. This contrasts with existing threshold-based detection systems that have difficulty in detecting slow-scanning worms because a small number of SYN packets generated by them do not cause visible traffic anomaly. Moreover, the FDF has low implementation complexity and the system parameters are independent on the network size and time-of-day. These features make it a promising method that can be deployed in any IDS systems and to run on a high-speed link.

### References