Detecting Internet Worms at Early Stage

Shigang Chen and Sanjay Ranka, *Fellow, IEEE*

Abstract—**Managing the security of enterprise networks has emerged to be a critical problem in the era of Internet economy. Arising as a leading threat, worms repetitively caused enormous damage to the Internet community during the past years. A new security service that monitors the ongoing worm activities on the Internet will greatly contribute to the security management of modern enterprise networks. This paper proposes an Internet-worm early warning system that automatically detects concerted scan activities and derives possible signatures of worm attacks. Its goal is to issue warning at the early stage of worm propagation and to provide necessary information for security analysts to control the damage. It reduces false positives by filtering out false scan sources. The system is locally deployable or can be codeployed amongst a group of enterprise networks. We provide both analytical and simulation studies on the responsiveness of this early warning system.**

Index Terms—**Early warning system, enterprise security management, Internet worm.**

I. INTRODUCTION

Motivation: Globalization does not happen only to the world's economies but also to the world's computer networks. By taking advantage of the universal connectivity offered by the Internet, the enterprise networks also face anonymous adversaries that may launch attacks from anywhere on the Internet. Today's enterprise security management must expand its scope to monitor the malicious activities on the Internet. Worm is an example.

Ever since the Morris worm showed the Internet community for the first time in 1988 that a worm could bring the Internet down in hours [1], new worm outbreaks have occurred periodically even though their mechanism of spreading was long well understood. Take a few examples. On July 19, 2001, the code-red worm (version 2) infected more than 250 000 hosts in just 9 hours [2]. Soon after, the Nimbda worm raged on the Internet [3]. On January 25, 2003, a worm called SQL-Slammer [4] caused widespread network congestion across Asia, Europe, and the Americas. Santy worm, W32/Zafi.D, variants of W32/Sober, variants of W32/MyDoom, variants of W32/Bagle, and W32/Sasser were reported last year. As recent as January, 2005, a MySQL UDF worm was reported at the U.S.-CERT web site.

The most common way for a worm to propagate is to exploit a security loophole in certain version(s) of a service software, which allows the worm to take control of the machine and copy itself over. For example, the Morris worm exploited a bug in *finger* and a trap door in *sendmail* of BSD 4.2 or 4.3, while the code-red worm took advantage of a buffer-overflow problem in the index server of IIS 4.0 or IIS 5.0. A worm-infected host scans the Internet for vulnerable systems. It chooses an Internet protocol (IP) address, attempts a connection to a service port (e.g., transmission control protocol (TCP) port 80 for code-red), and if successful, attempts the attack. The above process repeats with different random addresses. As more and more machines are compromised, more and more copies of the worm are working together to reproduce themselves. An explosive epidemic is therefore developed across the Internet.

Background: Most recent research on Internet worms concentrates on propagation modeling [5]–[8]. The defense against worms is still an open problem. There are currently few answers to the worm threat. One solution is to patch the software and eliminate the security defects [2]–[4]. That does not work because 1) software bugs seem always increase as computer systems become more and more complex and 2) not all people have the habit of keeping an eye on the patch releases. The patch for the security loophole that led to the SQLSlammer worm was released half a year before the worm appeared, and still tens of thousands of computers were infected. Intrusion detection systems and anti-virus software may be upgraded to detect and remove a known worm, but that happens only after the worm has spread and been analyzed.

For worms that propagate amongst certain type of servers, a solution is to block the servers' outbound connections so that the worms cannot spread among them. This approach works only when it is implemented for all or a vast majority of the servers on the Internet. Such an Internet-wide effort has not been and may never be achieved, considering that there are so many countries in the world and home users are setting up their servers without knowing this "good practice." In addition, the approach does not apply when a machine is used both as a server and as a client.

Moore *et al.* studied the effectiveness of worm containment technologies (*address blacklisting* and *content filtering*) and concluded that such systems must react in a matter of minutes and interdict nearly all Internet paths in order to be successful [8]. Williamson proposed to modify the network stack so that the rate of connection requests to distinct destinations is bounded [9], [10]. Schechter *et al.* used the sequential hypothesis test to detect scan sources and proposed a credit-based algorithm for limiting the scan rate of a host [11]. Weaver *et al.* developed containment algorithms suitable for deployment with high-speed, low-cost network hardware [12]. The main problem of the above approaches is that their effectiveness against worm propagation requires Internet-wide deployment. Gu *et al.* proposed a simple two-phase local worm victim detection algorithm based on both infection pattern and scanning pattern [13]. Chen and Tang designed a distributed anti-worm

architecture for Internet service providers (ISPs) to provide an anti-worm service to its customers [14].

Chen *et al.* proposed a sophisticated worm propagation model (called AAWP [15]) based on discrete times. In the same work, the model is applied to monitor, detect, and defend against the spread of worms under a rather simplified setup, where a range of unused addresses are monitored and a connection made to those addresses triggers a worm alert. The distributed early warning system by Zou *et al.* [16] also monitors unused addresses for the "trend" of illegitimated scan traffic on the Internet. There are two problems with these systems. First, the attackers can easily overwhelm such a system with false positives by sending packets to those addresses, or some normal programs may scan the Internet for research or other purposes and hit the monitored addresses. Second, to achieve good response time, the number of "unused addresses" to be monitored has to be large, but addresses are scarce resource in the IPv4 world, and only few has the privilege of establishing such a system. A monitor/detection system based on "used addresses" will be much more attractive. It allows more institutes or commercial companies to participate in the quest of defeating Internet worms.

Our contributions: An early warning system is essential in fighting against natural disasters such hurricanes, floods, wildfires, etc. Even for less predictable tornados or earthquakes, a just-in-time warning can be invaluable in saving lives and limiting damages. Similarly, in the Internet world, a worm early warning system is extremely important due to the enormous harm that a worm can potentially cause [5]. The warning system provides a new security service for protecting an enterprise network. It gives precious time for the enterprise security management team to react against an ongoing worm outbreak. Such a system is also practically feasible, given the worm's unique behavioral characteristics that have been well established, thanks to the flourish of recent research results in worm modeling.

In this paper, we propose WEW, an Internet-worm early warning system, which integrates a number of novel techniques that automatically detect concerted scan activities and derive possible signatures of worm attacks. The focus of this paper is on TCP-based worms. To make it a practical system, we have the following requirements. First, in order to minimize false positives, WEW must be resilient against counter measures and should be able to filter forged scan sources. Second, WEW should not depend on the ownership of a large segment of unused address space. Instead, its monitor/detection mechanisms should be all based on used address space. Third, in order to make the system efficient, WEW should commit very little or no computation/storage resources on normal traffic. Fourth, in order to make the system an integral part of Internet defense, WEW must assist security analysts and automated defense systems with information about the possible attack signatures, the worm-propagation speed, and the list of likely infected hosts.

Our contributions in meeting the above requirements are listed below.

- We propose worm-detection methods that do not require unused address space. Unlike the traditional approach that

keeps track of TCP SYN, we rely on TCP RESET packets to find the scan sources, which has greater accuracy and less overhead. Honeypots are used to capture the attack signatures of the scanning hosts, and the signatures can be presented to human analysts for confirmation or deployed on filtering devices or IDS systems.

- We propose an anti-spoof protocol that separates false scan sources from worm-infected hosts. The protocol covers various cases that may happen according to the TCP protocol. It remains stateless by using cookies, which makes it robust against denial-of-service (DoS) attacks.

- We provide an analytical study of the system and propose a new performance metric, *system sensitivity*, to capture the responsiveness of an early warning system in reporting an ongoing worm. We also present the simulation results of the system to support our claims.

The rest of this paper is organized as follows. Section II motivates the approaches that we use to separate worms from background activities. Section III proposes our Internet-worm early warning system. Sections IV and V present the analytical and simulation results, respectively. Section VI draws the conclusion.

II. MOTIVATION

A. Normal Activity Versus Worm Activity

We study the behavioral differences of a normal user and a worm-infected host. Research on worm detection is at its infancy. The scope of this paper is limited to TCP-based worms that employ random scanning to discover vulnerable systems. Most known worms belong to this category. UDP-based worms and targeted scanning worms [5] will be studied in our future work.

- A normal user accesses a server by its domain name, e.g., a URL for a web server. The domain name is resolved for an IP address via domain name system (DNS). If the domain name cannot be resolved, *no TCP connection will be made*. On the other hand, a worm attempts TCP connections to random addresses no matter whether these addresses are alive or not, which results in *a large number of connection failures*. Our experiment shows that over 99.6% of connections made to random addresses at TCP port 80 fails. The percentage will be even higher for less popular services.

- Comparing with worm-infected hosts that scan hundreds of addresses per second, normal users connect to different web servers at much slower rates due to manual operations and reading time.

- A user typically has a favorite server list (e.g., web sites). Those servers are visited most often and they are known to be up most of the time.

In summary, a worm-infected host will generate a persistent stream of failed connections, often at a high rate, while a normal user generates failed connections occasionally, at a much slower rate that does not persist. By observing the behavioral differences, we can distinguish normal users from worm-infected hosts.
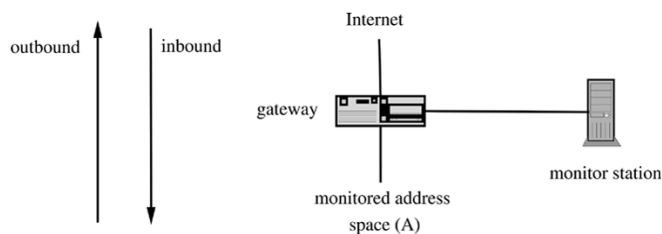
Fig. 1.   WEW: Sampling the Internet scan activities.

B. Scan and Worm

Address/port scan is part of worm activity. We study the differences between a "normal" scan and a worm scan.

- A scan is not necessarily accompanied by an attack. Its purpose may be simply to find out which addresses and which ports are alive. The number of scan sources on the Internet does not necessarily exhibit an increasing pattern during a short period of time (e.g., several days). Instead, it fluctuates up and down. On the other hand, the number of scan sources in a worm attack keeps increasing as more and more hosts are infected. It follows an exponential growth pattern.
- A normal scan typically probes destination addresses of the same subnet, whereas a worm scan typically probes the entire address space although it may emphasize more on the subnet where the infected host resides.

When a worm outbreak fully materializes, its presence is apparent even with the background of other scan activities, because tens of thousands of infected hosts behave similarly and their scan traffic can congest the networks around the world. On the other hand, when the worm propagation is at the early stage, its characteristics are buried among those of other scan sources. Consequently, to block the worm scan, we may have to block other scans together. This should be acceptable because any unauthorized scan activity is potentially harmful even if it is not related to a worm. Many other network-based attacks are preceded with a scan phase. Detecting all unauthorized external scans help to mitigate not only worms but also other threats.

III. INTERNET-WORM EARLY WARNING SYSTEM (WEW)

In this section, we propose an Internet-worm early warning system (WEW). We also describe how to filter false scan sources, identify persistent true scan sources, capture attack signatures, compose monitored address space, and deal with network address translation (NAT).

A. Monitoring Scan Sources

WEW is deployed at the main gateway of a large enterprise network. Its system architecture is shown in Fig. 1. The basic idea is to sample the Internet scan activities by monitoring a portion of the IPv4 address space behind the gateway. The system detects potential worm outbreak by analyzing the pattern of increase in external scan sources and comparing their similarity. It captures the common signature from those sources in order to assist human analysis or automatically reconfigure a filtering device to block them.

Let A be the monitored address space, which is separated from the rest of the Internet by a gateway. The main functions of WEW are implemented by the gateway and a monitor station. For the purpose of simplicity, our description assumes a single gateway. If an enterprise network is multihomed, our system should be deployed at all gateways that connect the network to ISPs, and all gateways will send their reports to the same monitor station or multiple stations for improved performance and resiliency.

In the text of this paper, the gateway is the default place for implementing certain functions (to be described). In the actual deployment, all these functions can move to a dedicated device such as a firewall behind the gateway. This does not change the fundamentals of the proposed defense system but allows the flexibility of adding resources to handle a large customer.

The primary task of WEW is to profile all external scan sources from the Internet. To do so, a naive approach is to keep track of all inbound TCP SYN packets. If the number of SYN packets from an external host exceeds a threshold value within a period of time, the host is thought to be scanning. Although this approach is commonly used in commercial intrusion-detection systems, it is neither accurate nor efficient. The number of TCP connections through a major router can be hundreds of thousands per second. Monitoring all TCP SYN incurs significant overhead because it spends computation/storage resources on all TCP connection attempts, even though only a small fraction may come from scan sources.

A better approach is to monitor *outbound* TCP RESET packets, which indicate failed *inbound* connection attempts, as illustrated in Fig. 2, where the worm scans for and then infects certain types of web servers. A connection attempt fails if the destination host does not exist or the destination port is not open. Specifically, if a SYN packet is sent to an existing host with the destination port closed, a TCP RESET packet will be returned; if a SYN packet is sent to a nonexisting host, an ICMP host-unreachable packet is returned. Servers of certain type are likely to occupy only a small portion of the IPv4 address space. For example, the web servers occupy roughly 0.4% of the IPv4 address space according to our scan experiment. Consequently, most random connections made by a worm scan will fail, which also implies that the scan rate can be roughly measured by the rate of failed connections. As we argued in the previous section, a normal user does not persistently cause connection failures to different destination addresses at a high rate. Therefore, by monitoring TCP RESET and ICMP host-unreachable packets, we can set worm scan sources apart from normal users.

Assume the monitored address space A is densely populated. Outbound TCP RESET packets will be the main form of response to failed inbound connections. Examining RESET packets alone at the gateway will suffice the detection of external worm scan sources. Specifically, when WEW detects the number of RESET packets to an external host exceeds a threshold value within a period of time, it reports the host as a likely scan source. We address the case of sparsely populated A in Section III-F.

B. Identifying False Scan Sources

An adversary may stimulate WEW to report false positives by generating false scan sources (Fig. 3). Moreover, if the number
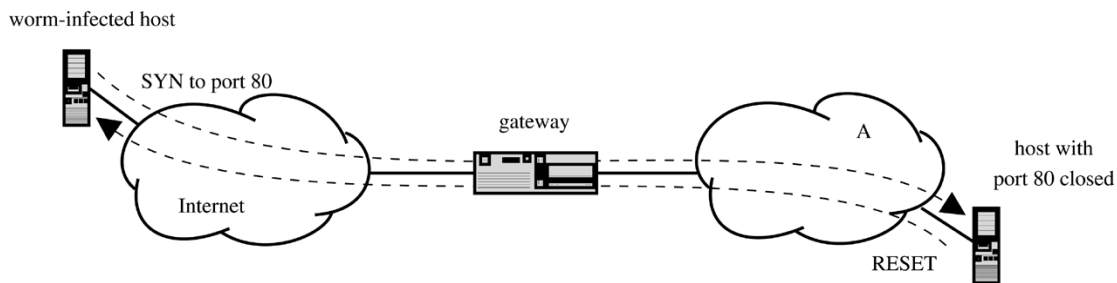
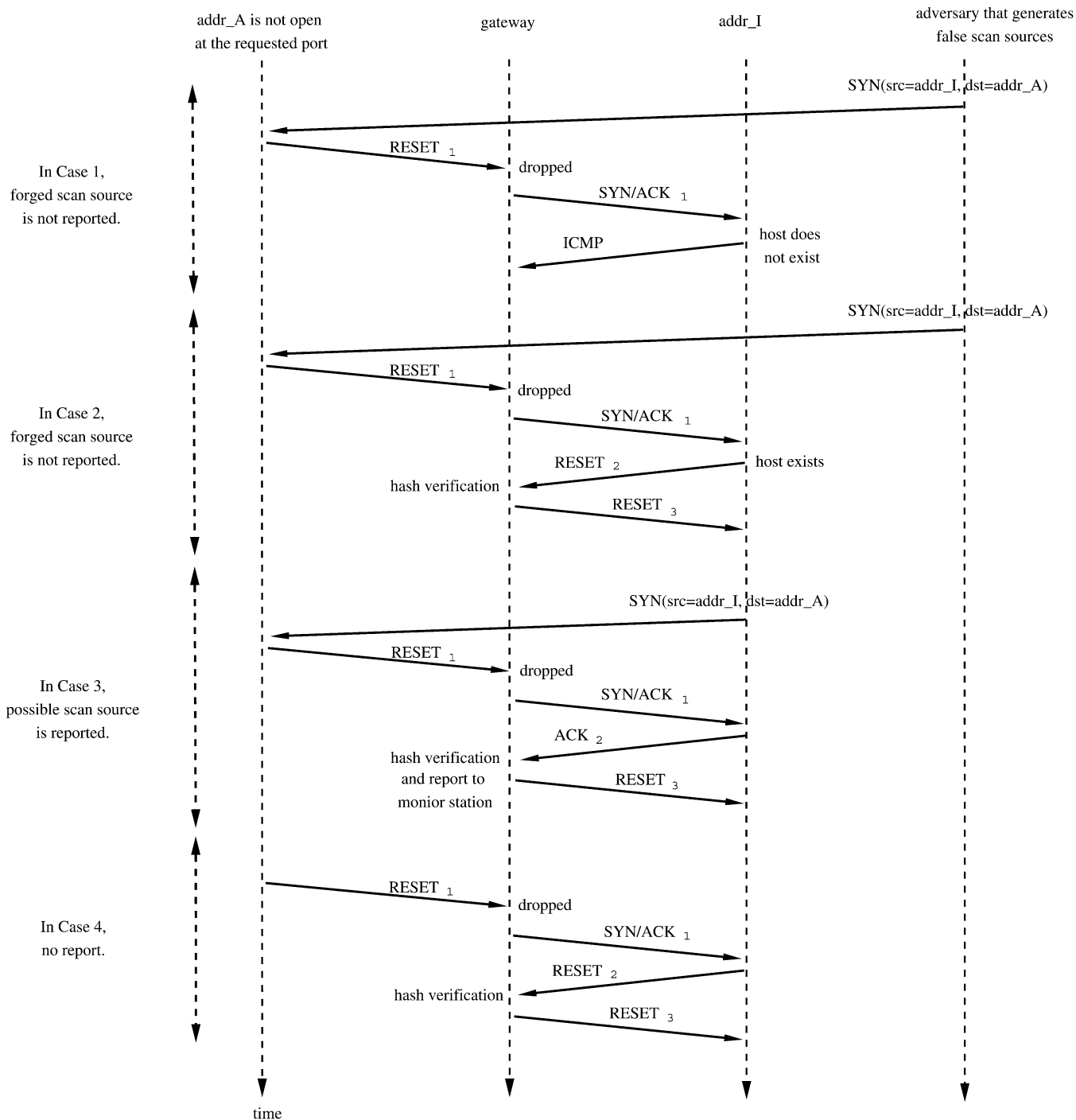Fig. 2.   Failed connection attempt.

Fig. 3.   Anti-false scan sources.

of false sources is overwhelmingly large, it can degrade the performance of the monitor system or even constitute a denial-of-server attack.

As an example, a hostile host may cause a false positive by simulating a worm attack. It starts with connections to random destination addresses from a single forged source address. As time passes, it makes random connections from more and more forged source addresses with the total number growing exponentially. A system that monitors failed connections will detect a worm-like increase in scan sources, and consequently, raise a false alarm.

We propose the following solution to the above problem. Consider a connection attempt from an external host from the Internet to an internal host in A. Let $addr\_I$ and $port\_I$ be the address and the port of the Internet host, respectively. Let $addr\_A$ and $port\_A$ be the address and the port of the host in A, respectively. Let $ackNum$ and $seqNum$ be the acknowledgment number and the sequence number of a TCP segment, respectively. Let $key1$ and $key2$ be two secrets known only by the gateway. To distinguish different messages of the same type, we use a subscribe number after the type.

When the gateway receives an outbound TCP RESET_1 packet from A to the Internet, it neither notifies the monitor station immediately nor forwards the packet. Instead, it constructs a TCP $\mathrm{SYN/ACK}_1$ packet with $addr\_I$, $addr\_A$, $port\_I$, $port\_A$, and $ackNum$ copied from the RESET_1 packet, and with the other header fields properly set. In particular, $seqNum$ consists of two parts: a b-bit random number, denoted as $seqNum.rand$, and a $(32-b)$-bit keyed hash, $hash(addr\_I \mid addr\_A \mid port\_I \mid port\_A \mid seqNum.rand, key1)$, denote as $seqNum.hash$, where "|" is the concatenation operator. The keyed hash (e.g., SHA1 or MD5) serves as the authentication code for the concatenation of the source/destination address/port. The purpose of $SeqNum.Rand$ is to make the hash result different each time even with the same source/destination address/port. The above constructed $\mathrm{SYN/ACK}_1$ packet is sent out, and the RESET_1 packet is dropped by the gateway. The possible outcomes from the $\mathrm{SYN/ACK}_1$ are listed below. In the first three cases, the RESET_1 packet is caused by a SYN_1 packet from the Internet that initiates a connection to a host in A.

Case 1) If the initial SYN_1 packet carries a forged source address that does not exist, then $\mathrm{SYN/ACK}_1$ will result in an ICMP host-unreachable packet being returned.

Case 2) If the initial SYN_1 packet carries a forged source address that exists, then $\mathrm{SYN/ACK}_1$ will result in a RESET_2 packet being returned.

Case 3) If the initial SYN_1 packet is from a real source, then $\mathrm{SYN/ACK}_1$ will result in an ACK_2 packet being returned to complete the connection.

Case 4) If the dropped RESET_1 was not in response to a SYN packet, then $\mathrm{SYN/ACK}_1$ is not expected by its receiver and thus a RESET_2 packet will be returned.

For Case 1), the result is the same as if RESET_1 was sent. The gateway does not need to do anything further. For Case 3), the gateway should verify the keyed hash, which is now in the acknowledge field. If the verification is successful, it resets the connection and reports *a failed-connection attempt* (by $addr\_I$)

to the monitor station. The gateway cannot distinguish Case 2) and Case 4) because it receives the same RESET_2 packet in both cases. It treats them in the same way by sending back a RESET_3 after successfully verifying the keyed hash. Most fields in RESET_3 can be copied from RESET_2. Note that RESET_3 is desirable in Case 4) because the previous RESET_1 might have been sent by $addr\_A$ to terminate an existing TCP connection with $addr\_I$ but RESET_1 was dropped by the gateway. In this scenario, RESET_3 allows $addr\_I$ to terminate the connection as wished.

Next, we give additional implementation details on how the gateway separates the real scan sources from the false ones and report only the former. For each RESET or ACK packet received from the Internet, the gateway performs a hash verification on $ackNum$, which is copied from $seqNum$ if the packet is the response to $\mathrm{SYN/ACK}_1$. Based on the same bit positions of $seqNum.rand$ and $seqNum.hash$, we define $ackNum.rand$ and $ackNum.hash$, which together form the 32-bit $ackNum$. The gateway tests if $ackNum.hash = hash(srcAddr \mid dstAddr \mid srcPort \mid dstPort \mid ackNum.rand, key1)$, where $srcAddr$, $dstAddr$, $srcPort$, and $dstPort$ are the source address, destination address, source port, and destination port of the packet, respectively. If the hash verification is not successful, the gateway forwards the packet as normal. If the verification is successful, the packet must be the response to a previously constructed $\mathrm{SYN/ACK}_1$ (for Cases 2–4). The gateway does one of the following two actions.

1) If the packet is an ACK_2 (i.e., Case 3), the gateway replies back a RESET_3 to terminate the connection, reports a failed-connection attempt to the monitor station, and drops the ACK_2.

2) If the packet is a RESET_2 (i.e., Cases 2 or 4), the gateway replies back a RESET_3 and drops the RESET_2.

It is possible for an arbitrary ACK packet to satisfy the condition, $ackNum.hash = hash(srcAddr \mid dstAddr \mid srcPort \mid dstPort \mid ackNum.rand, key1)$. But the possibility for that to happen is only $1/2^{32-b}$. Because b does not need to be large (e.g., $b \le 8$), this probability is extremely small. If it does happen, a normal TCP connection may be mistakenly terminated. This rare problem can be solved with a little extra overhead: The gateway forwards the ACK packet after successful hash verification and waits for a response for the TCP timeout period. If a RESET packet comes back from the receiver of the ACK, the gateway performs the above action 1); otherwise, it does nothing.[1]

A worm-infected host is typically open at the port that it attacks. When a gateway reports a failed connection made by an external host $addr\_I$ to a destination port $port\_A$, it may perform an extra test to see if $addr\_I$ itself is open at the same port. It sends the host $addr\_I$ a constructed SYN_3 packet with the destination address/port being $addr\_I$ and $port\_A$, the source address (denoted as $addr\_*$) being randomly selected from A, the source port (denoted as $port\_*$) also randomly selected, and the sequence number being $seqNum.rand \mid hash(addr\_I \mid addr\_* \mid port\_A \mid port\_*, seqNum.rand, key2)$. Note that

---

[1]If the receiver does not have a TCP connection for the ACK packet, the RESET packet will come back immediately, which is captured by the gateway. This RESET packet is *expected by the gateway* and should not trigger $\mathrm{SYN/ACK}_1$ to be constructed.

the gateway remains stateless after this constructed packet is sent. For each received SYN/ACK, the gateway tests if *ackNum* satisfies $ackNum.hash = hash(srcAddr \mid dstAddr \mid srcPort \mid dstPort \mid ackNum.rand, key2)$. If so, the gateway drops the SYN/ACK, replies with a RESET_4 to terminate the connection, and reports to the monitor station that the potential scan source *srcAddr* is also open at the port it scans (a sign of worm activity).

The gateway may rely on an application specific integrated circuit (ASIC) chip to construct the artificial SYN/ACK_1 and SYN_3 and to perform the hash verification, or it may send a special log to a dedicated machine, which performs such tasks. The hash functions (MD5 or SHA1) are very efficient and there are software/hardware products operating at the line speed of Gigabit per second, which prevents it from being a bottleneck.

One may argue that an attacker may stimulate WEW to send SYN/ACK_1 and RESET_3 packets to an innocent host for a DoS attack. It will become clear shortly that the number of SYN/ACK_1 and RESET_3 sent by WEW to an external address is limited because WEW stops sending these packets after the address is placed into a so-called scan list. Furthermore, WEW may be configured such that, after a worm warning is issued and confirmed, it ceases further operations in order to relieve the network that is already stressed by the worm traffic.

C. Identifying Persistent Scan Sources

Based on the connection-failure reports from the gateway, the monitor station generates a list of possible scan sources, called *scan list* and denoted as Ω. It consists of external addresses x that have attempted *failed connections* to more addresses in A than a threshold k during the day, where k is a user-configured value. The gateway does not need to continue investigating the failed connections made from addresses currently in Ω; it forwards RESET_1 packets to those addresses without performing the operations described in the previous section.

Some external hosts may temporarily behave abnormally, resembling a scan source. In order to release the addresses that are mistakenly placed into the scan list, each address in the list is associated with a timer, which is reset to a user-configured value T at the beginning of each day. The address is removed from the list after timeout. Note that after an address is removed from Ω, its information is still logged. Every time an address is reinserted into the list because of k more failed connections, its timer is doubled. Therefore, normal users with temporary abnormal behavior will be removed from the scan list, while persistent scanning sources will remain in the list.

Ω provides the list of likely infected hosts. When a worm threat is confirmed, the security administrator may decide to block all inbound connections initiated from the addresses in Ω. When blocking is activiated, the maximum scan rate that a worm-infected host may achieve on A will be bounded. Let D be the time of a day and L be the maximum number of times that an address can be inserted and removed from Ω during a day. We have

$$\Sigma_{i=1}^{L} 2^{i-1}T \leq D$$
$$L \leq \log_2\left(\frac{D}{T} + 1\right). \tag{1}$$

An external infected host will be placed in Ω each time after k failed connection attempts. Since most random connection attempts by the infected host will fail, the scan rate $\overline{r}$ that the host can achieve on A must be bounded

$$\overline{r} \leq \frac{kL}{D} = \frac{k\log_2\left(\frac{D}{T}+1\right)}{D}.$$

For example, if $k = 10$ and $T = 1$ min, then a worm-infected host can scan A at a rate bounded by $1.2 \times 10^{-3}/\text{s}$.

D. Capturing Attack Signatures

For scan sources in Ω, the gateway may be configured to redirect some of their connection attempts to one or multiple honeypots,[2] which complete the connections and record the session data. A string-match algorithm is used to find the common signatures among the recorded sessions. The most-frequent signatures among all or some scan sources are potentially the attack signatures, which are presented to the security-management team for further analysis.

To assist the honeypots, the gateway maintains a redirection table. Let *addr_H* and *port_* be the address and the service port of a honeypot. Consider an inbound SYN packet from an external host to an internal host with *addr_ I/port_I* and *addr_A/port_A* being the source address/port and destination address/port. Suppose *addr_I* belongs to Ω, and the SYN is selected by the gateway for redirection. The gateway creates a new entry in the redirection table, denoted as (*addr_ I, port_ I, addr_A, port_A*). It then substitutes the destination address/port by *addr_ H/port_H* and forwards the packet to the honeypot. When the gateway receives an inbound packet that is not a SYN, it looks up the redirection table for an entry that matches the packet's source address/port. If there is one, the destination address/port will be substituted by *addr_H/port_H*. Otherwise, the packet is processed as normal. When the gateway receives an outbound packet from the honeypot, it looks up the table for an entry whose *addr_I/port_I* match the packet's destination address/port, and then it substitutes the packet's source address/port with *addr_A/port_A* of the entry. An entry in the redirection table will be removed when the appropriate FIN or RESET packets are received, or there is no match for the entry during a period that will timeout a TCP connection.

Because a worm attack requires the compromise of a remote host and then the transfer of the worm code to the compromised host, the focal point of signature generation should be the payload of the session packets recorded by the honeypot. For a *standard worm*, different attack sessions carry identical payload content. For a *polymorphic worm*, different attack sessions may carry payload that is slightly different. String-match algorithms are used to find the longest substrings that are common to a set of similar sessions. The common substrings or their hash values can be used as the traffic signature for those sessions.

When a worm attack is confirmed, the signatures computed by the monitor station can be immediately deployed on the gateway and the internal firewalls. For a standard worm, a hash-based signature is used. An attack session may consist of multiple inbound packets. Hence, the hash algorithm must support progressive computation based on partial payload. Examples of such algorithms are MD5 and SHA1. For a

---

[2]They may be hardened replicates of local systems to be protected.

polymorphic worm, the common substrings are used, and the inbound packets of a session are progressively matched against those substrings.

E. Issuing Worm Warning

If more and more scan sources are detected and inserted in Ω and the number of those sharing the same or similar signatures exceeds a threshold n_0, then warning of a possible worm attack will be issued together with the identified signatures. Although the process of issuing a warning is likely to be semiautomatic with the human investigators making the final approval, the WEW system greatly improves the timeliness and accuracy of the process.

While the focus is on monitoring the external scan sources, WEW can simultaneously monitor the internal scan sources. When the internal hosts probe random addresses outside of A, their scan activities are profiled by the gateway. More specifically, after random SYN packets are sent to the Internet, the RESET packets will come back to the gateway, which performs the same techniques as described in Section III-B to filter the false scan sources and only record those internal sources that pass the hash verification. When a worm outbreak is confirmed, the internal scan list created by WEW presents the system administrator with valuable information about which hosts may have already been compromised.

The deployment of WEW does not have to be kept secret. A worm may evade the detection by not scanning the networks that deploy WEW. In this case, WEW cannot issue an early warning, but it protects the networks from being infected by the worm.

F. NAT and Sparsely Populated Address Space

Suppose an enterprise network wants to deploy WEW system but it has implemented NAT on its gateway. Connections are allowed to be initiated from the internal hosts to the Internet, but are not allowed to be initiated from the Internet to the internal hosts except for a list of public servers. More specifically, the gateway will drop all inbound SYN packets destined to an address in A that is not assigned to a public server. Because the SYN does not reach an actual host, no RESET will be replied. Recall that WEW relies on RESET packets to work. To solve this problem, for a gateway that implements NAT, when it drops an inbound SYN packet, it must immediately create SYN/ACK_1 based on the information from the SYN, and then follow the same protocol described in Section III-B.

A different problem is associated with a sparsely populated address space. WEW works better for a densely populated address space, where RESET is the main form of response to failed connection attempts. If most addresses in A are not alive, then ICMP host-unreachable packets, instead of TCP RESET, are replied for most failed connection attempts. ICMP host-unreachable packets do not carry the information that is needed to create SYN/ACK_1. To solve this problem, the gateway must be configured with a list of unused address prefixes, which is matched against the inbound SYN packets. If there is a match, the SYN is dropped and SYN/ACK_1 is created.

G. Composing the Monitored Address Space

WEW works well for a large enterprise network, but not so well for a small enterprise because the system sensitivity in worm detection is directly related to the size of the monitored address space (Section IV). To solve this problem, small enterprise networks can join together to form a large aggregated address space A. Each participating enterprise deploys WEW on its gateway and a local monitor station. The local monitor station periodically reports its scan list and the most-frequent traffic signatures from the listed scan sources, digitally signed by a private key, to a trusted WEW authority, which performs forensic analysis on the scan lists and the traffic signatures from all participating enterprises. If the network bandwidth and the security policy allow, the report should also include the summary of the new failed connections recorded by the local monitor station since the previous report to the authority. The WEW authority can be elected from the participating enterprises. In case that multiple or all enterprises want to be the authorities, each enterprise sends its periodical report to all authorities. The WEW authority may also be a trusted third party that provides a registration based service to coordinate the participating enterprises.

IV. ANALYSIS

A. Basics

Worm propagation can be roughly characterized by the classical simple epidemic model [5], [8], [17]

$$\frac{di(t)}{d(t)} = \beta i(t)(1 - i(t)) \tag{2}$$

where $i(t)$ is the percentage of vulnerable hosts that are infected with respect to time t, and β is the rate at which a worm-infected host detects other vulnerable hosts. It can be proved that $\beta = r(V/N)$ [8], [14], where r is the rate at which an infected host scans the Internet, N is the size of the IPv4 address space, and V is the total number of vulnerable hosts. Suppose the worm starts with one infected host at $t = 0$. Solving (2), we have

$$i(t) = \frac{e^{r(V/N)(t-c)}}{1 + e^{r(V/N)(t-c)}}$$

where $c = -(N/rV)\ln(1/V - 1)$. The number of hosts that are infected at time t is

$$n(t) = V\frac{e^{r(V/N)(t-c)}}{1 + e^{r(V/N)(t-c)}}. \tag{3}$$

The time it takes for n vulnerable hosts to be infected is

$$t(n) = \frac{N}{r \cdot V} \ln \frac{n}{V - n} + c. \tag{4}$$

In reality, worms propagate slower than the above equation due to a number of factors. First, once a large number of hosts are infected, the aggressive scanning activities often cause widespread network congestions, and consequently, many scan messages are dropped. Second, when a worm outbreak is announced, many system administrators shut down vulnerable servers or remove the infected hosts from the Internet. Third,

some types of worms enter dormant state after being active for a period of time. Due to the above reasons, code-red spread much slower than the calculation based on (4). A more sophisticated model that considers the first two factors was proposed in [7], which fits better with the observed code-red data. Additional modeling work can be found in [6], [15]. Our following analysis based on (4) can be considered as the lower bound on the performance of the WEW system.[3]

B. System Sensitivity

Suppose WEW observes an increasing number of scan sources sharing a similar traffic signature, and it generates a worm warning when the number of similar scan sources reaches n_0, which is a preconfigured parameter. Before we proceed with the analysis, we want to stress that, although widespread scan activity is not necessarily due to a worm attack, large-scale coordinated scan effort warrants a checkout. The progressive increase in the scan sources points toward a *likely* worm attack.

A new performance metric, called *system sensitivity*, is defined as (n_w/n_0), where n_w is the number of infected hosts on the Internet at the time when the warning is issued. The system sensitivity indicates how quickly WEW detects a worm attack and, more specifically, it measures the gap between the actual number of infected hosts and the targeted goal (n_0) when a warning is issued. Ideally, we want the system sensitivity to be close to one.

Consider a monitored address space A. By (4), the time it takes for a worm to infect n_0 hosts is

$$t(n_0) = \frac{N}{r \cdot V} \ln \frac{n_0}{V - n_0} + c.$$

Among the n_0 hosts, let x be the last infected one. Let Δt be the time it takes for WEW to insert x into the scan list after x is infected. x attempts $r\Delta t$ random connections during the period of Δt. Among those connections, $r\Delta t(A/N)$ are made to A. Let α be the percentage of connections that fail and are reported by the gateway(s). As we discussed previously, α is likely to be large if A is densely populated or the gateway(s) are configured to catch SYNs that are sent to unreachable addresses. x is inserted into the scan list when $\alpha r \Delta t(A/N) = k$. We have

$$\Delta t = \frac{kN}{\alpha r A}.$$

A warning is subsequently generated, which happens at time $t(n_0) + \Delta t$, ignoring the report transfer delay from the gateway to the monitor station and the processing delay at the monitor station, also ignoring the transfer delay and the processing delay of the WEW authority if one is used (Section III-G). Therefore, the *response time* for WEW to warn an ongoing worm is about

$$t(n_0) + \Delta t = \frac{N}{r \cdot V} \ln \frac{n_0}{V - n_0} - \frac{N}{r \cdot V} \ln \frac{1}{V - 1} + \frac{kN}{\alpha r A}$$
$$= \frac{N}{r \cdot V} \ln \frac{n_0(V - 1)}{V - n_0} + \frac{kN}{\alpha r A}.$$

---

[3]It means the system is likely to perform better in reality.

At the time when the warning is issued, the actual number of infected hosts is about

$$
\begin{aligned}
n_w &= n(t(n_0) + \Delta t) \\
&= V \frac{e^{r(V/N)(t(n_0)+\Delta t-c)}}{1 + e^{r(V/N)(t(n_0)+\Delta t-c)}} \\
&= V \frac{n_0 e^{(kV/\alpha A)}}{V - n_0 + n_0 e^{(kV/\alpha A)}} \\
&\le n_0 e^{(kV/\alpha A)}.
\end{aligned}
$$

Therefore, we have an upper bound on system sensitivity

$$\frac{n_w}{n_0} \le e^{(kV/\alpha A)}. \tag{5}$$

Interestingly, the system sensitivity is independent of the scan rate of a worm. A sufficient condition to achieve a target sensitivity of γ is

$$A \le \frac{kV}{\alpha \ln \gamma}. \tag{6}$$

Suppose we have a target sensitivity of 3. If $V = 10^6$, $k = 10$, and $\alpha = 0.8$, then by (6) it is sufficient that $A = 1.14 \times 10^7 < 2^{24}$, which is a Class A network. Because addresses can be used ones, a large ISP or the core of a research network can achieve decent sensitivity with WEW. Small enterprise networks can cooperate to form a large address space.

V. SIMULATION RESULTS

This section presents the simulation results of the proposed Internet-worm early warning system. The default simulation parameters are described as follows. There are $V = 10^6$ vulnerable servers on the Internet of size $N = 2^{32}$. WEW is deployed on an aggregated address space $A = 2^{24}$. Suppose 80% of failed connection attempts are reported by the gateway(s). An external address is inserted in the scan list after it is detected to have made failed connections to $k = 10$ different internal addresses; recall that only persistent scan sources will keep staying in the scan list (Section III-C). WEW issues a worm warning when it detects at least $n_0 = 300$ scan sources with similar traffic signatures. Although the target number is n_0, at the time when the warning is issued, the actual number n_w of infection on the Internet will be larger, which was explained in Section IV. The default parameters are always assumed unless the figures indicate otherwise.

Fig. 4 shows the system sensitivity with respect to n_0 and A. The system sensitivity decreases with respect to n_0. For example, with $A = 2^{24}$, the sensitivity is around 2.1, decreasing only slightly as n_0 increases. On the other hand, it increases significantly when A decreases. Hence, WEW works better when it monitors a larger address space. Fig. 5 shows that n_0 affects n_w sublinearly (almost linearly for large A values). Even when A is relatively small and, thus, the sensitivity value is big, as long as n_0 is small, then n_w will still be small and account for only a small percentage of V. For example, if $A = 2^{22}$ and $n_0 = 300$, n_w accounts for just 0.4% of V.

Fig. 6 shows the time it takes WEW to report an ongoing worm attack, with respect to the worm scanning rate r and n_0. For comparison, we also plot $t(250\,000)$, the time it takes the
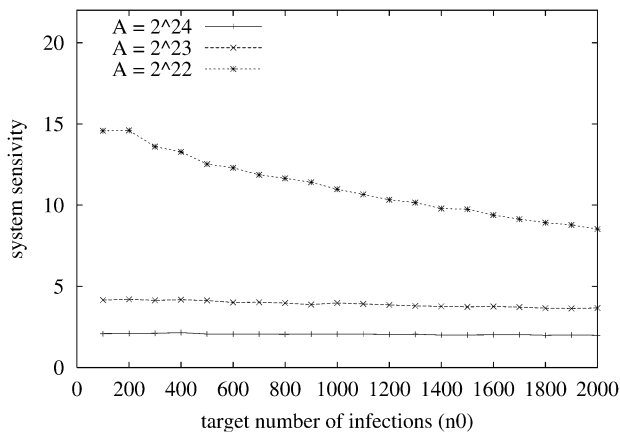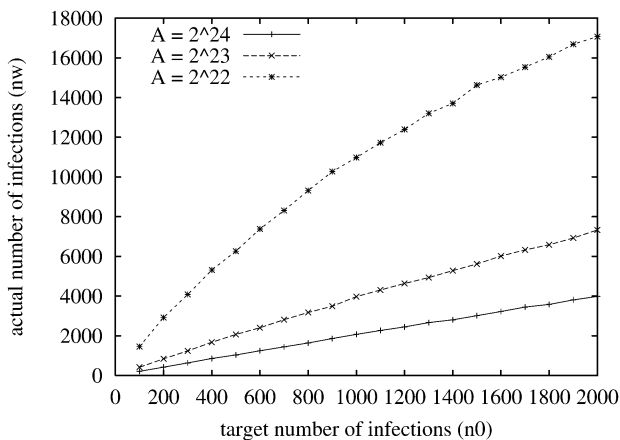
Fig. 4.   System sensitivity.
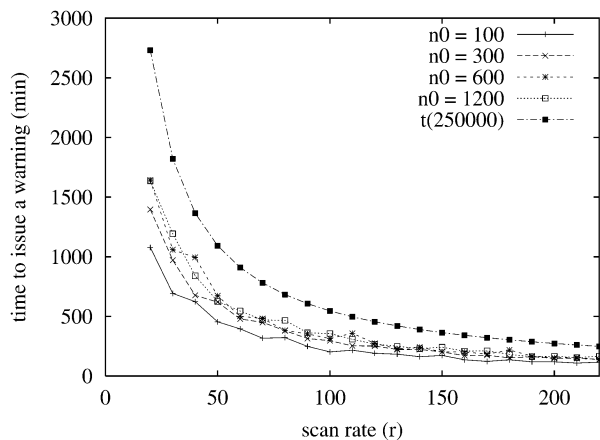

Fig. 5.   Actual number of infections.

Fig. 6.   System response time.

worm to infect a quarter of all vulnerable hosts. Fig. 7 shows the actual number of hosts that are infected by the worm at the time when the warning is issued. The code red took about 9 hours to infect $250\,000$ hosts [2], which roughly corresponds to the data point at $r = 100/\min$. The figures show that, when $n_0 = 300$, it takes WEW about 296 minutes to issue a warning, just
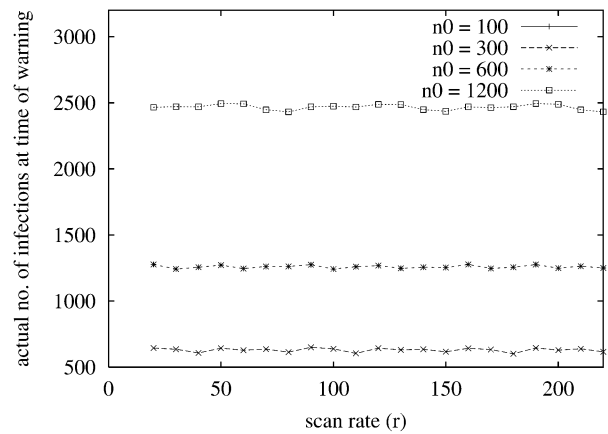

Fig. 7.   Actual number of infections.

before the fifth hour from the start of the worm attack and when only 639 hosts are infected. When WEW issues the warning, it will also give a scan list that contains many of those infected hosts. When an automated (or semiautomated) defense system is introduced in the future, WEW can be a valuable component that activates the defense measures and provides information about the attack traffic and the hosts that need to be disinfected.

VI. CONCLUSION

This paper proposes an Internet-worm early warning system (WEW), designed as a component for a future worm defense system. Its goal is to issue a warning about a worm attack before it is fully propagated across the Internet. It is also able to provide information about the attack signature and the infected hosts. WEW detects scan sources by examining TCP RESET packets. Novel techniques are developed to remove false scan sources, which reduces false positives. WEW is evaluated through both analysis and simulations, which demonstrate the practicality of the system.

REFERENCES

[1] J. Rochlis and M. Eichin, "With microscope and tweezers: The worm from MIT's perspective," *Commun. ACM*, vol. 32, no. 6, pp. 689–698, Jun. 1989.

[2] (2001) CERT advisory CA-2001-23 "code red" worm exploiting buffer overflow in IIS indexing service DLL. Computer Emergency Response Team. [Online]. Available: http://www.cert.org/advisories/CA-2001-23.html

[3] (2001) CERT advisory ca-2001-26 Nimda worm. Computer Emergency Response Team. [Online]. Available: http://www.cert.org/advisories/ca-2001-26.html

[4] (2003) CERT advisory CA-2003-04 MS-SQL server worm. Computer Emergency Response Team. [Online]. Available: http://www.cert.org/advisories/CA-2003-04.html

[5] S. Staniford, V. Paxson, and N. Weaver, "How to own the Internet in your spare time," in *Proc. 11th USENIX Security Symp.*, San Francisco, CA, Aug. 2002.

[6] M. Liljenstam, Y. Yuan, B. Premore, and D. Nicol, "A mixed abstraction level simulation model of large-scale Internet worm infestations," in *Proc. 10th IEEE/ACM Symp. Modeling, Analysis, Simulation Comput. Telecommun. Syst.*, Oct. 2002, pp. 109–116.

[7] C. C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *Proc. 9th ACM Conf. Comput. Commun. Security*, Nov. 2002.

[8] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proc. 22nd Annu. Joint Con. IEEE Comput. Commun. Soc.*, San Francisco, CA, Mar. 2003, pp. 1901–1910.

[9] M. M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," in *Proc. Annu. Comput. Security Applicat. Conf.*, Dec. 2002, pp. 61–68.

[10] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proc. 12th USENIX Security Symp.*, Washington, DC, Aug. 2003.

[11] S. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Proc. 7th Int. Symp. Recent Advances in Intrusion Detection*, Sophia Antipolis, French Riviera, France, Sep. 2004.

[12] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *Proc. 13th USENIX Security Symp.*, San Diego, CA, Aug. 2004.

[13] G. Gu, D. Dagon, X. Qin, M. I. Sharif, W. Lee, and G. F. Riley, "Worm detection, early warning, and response based on local victim information," in *Proc. 20th Annu. Comput. Security Applicat. Conf.*, Tucson, AZ, Dec. 2004.

[14] S. Chen and Y. Tang, "Slowing down Internet worms," in *Proc. 24th IEEE Int. Conf. Distrib. Comput. Syst.*, Tokyo, Japan, Mar. 2004, pp. 312–319.

[15] Z. Chen, L. Gao, and K. Kwiat, "Modeling the spread of active worms," in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Soc.*, San Francisco, CA, Mar. 2003, pp. 1890–1900.

[16] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for Internet worms," in *Proc. 10th ACM Conf. Comput. Commun. Security*, Washington, DC, Oct. 2003.

[17] H. W. Hethcote, "The mathematics of infectious diseases," *SIAM Rev.*, vol. 42, no. 4, pp. 599–653, 2000.

[18] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, Jun. 1977.
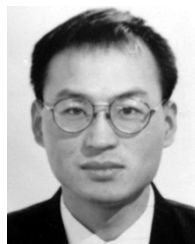
Shigang Chen received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, Urbana, in 1996 and 1999, respectively.

After graduation, he had worked with Cisco Systems for three years before joining the University of Florida, Gainesville, as an Assistant Professor in 2002. His research interests include network security, quality-of-service, and sensor networks.

Sanjay Ranka (S'87–M'88–SM'01–F'02) received the B.Tech. degree in computer science from the Indian Institute of Technology (IIT), Kanpur, India, in 1985 and the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, in 1988.

He is a Professor in the Department of Computer Science, University of Florida, Gainesville. He is a world-renowned technologist in the areas of large-scale software systems, high-performance networking, interactive marketing, CRM, data mining, and optimization. Most recently, he was the Chief Technology Officer at Paramark, where he developed real-time optimization software for optimizing marketing campaigns. He has also held positions as a tenured faculty member at Syracuse University and as a Researcher/Visitor at IBM T. J. Watson Research Laboratories and Hitachi America Limited. He has coauthored *Elements of Neural Networks* (Cambridge, MA: MIT Press) and *Hypercube Algorithms* (New York: Springer-Verlag). He is an Advisory Board Member of the IEEE Technical Committee on Parallel Processing and a member of IFIP Committee on System Modeling and Optimization.