SIMULATION

Enhancing SWORD to Detect Zero-Day-Worm-Infected Hosts

Shad Stafford, Jun Li and Toby Ehrenkranz

The online version of this article can be found at:

http://sim.sagepub.com/cgi/content/abstract/83/2/199

Published by:

$SAGE Publications

http://www.sagepublications.com

On behalf of:

$$\boxed{\text{SCS}}$$

Society for Modeling and Simulation International (SCS)

Additional services and information for *SIMULATION* can be found at:

Email Alerts: http://sim.sagepub.com/cgi/alerts

Subscriptions: http://sim.sagepub.com/subscriptions

Reprints: http://www.sagepub.com/journalsReprints.nav

Permissions: http://www.sagepub.com/journalsPermissions.nav

Enhancing SWORD to Detect Zero-Day-Worm-Infected Hosts

Shad Stafford
Jun Li
Toby Ehrenkranz
Department of Computer Science
University of Oregon
Eugene
OR 97403-1202, USA
staffors@cs.uoregon.edu

Once a host is infected by an Internet worm, prompt action must be taken before that host does more harm to its local network and the rest of the Internet. It is therefore critical to quickly detect that a worm has infected a host. In this paper, we enhance our SWORD system to allow for the detection of infected hosts and evaluate its performance. This enhanced version of SWORD inherits the advantages of the original SWORD: it does not rely on inspecting traffic payloads to search for worm byte patterns or setting up a honeypot to lure worm traffic. Furthermore, while acting as a host-level detection system, it runs at a network's gateway and stays transparent to individual hosts. We show that our enhanced SWORD system is able to quickly and accurately detect if a host is infected by a zero-day worm. Furthermore, the detection is shown to be effective against worms of different types and speeds, including polymorphic worms

Keywords: Internet worms, worm detection, net-work security, host infection

1. Introduction

The launching of a worm can have disastrous effects on millions of computers on the Internet in just a few short minutes [2, 3], potentially disrupting the operation of critical services such as emergency call centers [4]. The cost of disrupted service and repair from worms can also be extremely high; for example, it is estimated that the costs associated with the Code Red and Sapphire/Slammer worms are over US$ 3 billion. Perhaps even more alarming, researchers have found that none of the worm attacks so far have come close to causing the amount of damage they are capable of [5].

With worms and their destructiveness gaining widespread recognition, it seems only a matter of time until new worms are created with even higher rates of spread. Worms that take advantage of a heretofore unknown vulnerability, so-called *zero-day* worms, are particularly dangerous because many existing security techniques require

prior knowledge of the exploit in order to detect and defeat it. These worms could become even more dangerous by creating their own coordinated networks from the infected hosts [6]. In order to achieve effective containment, as shown in [7], reaction times must be on the order of a few minutes or less. Any worm defense mechanism which requires human action, therefore, is just not feasible.

Our recent research has devised an approach, named SWORD [8], to detect the occurrence of zero-day worms at an administrative domain level. SWORD has a number of desirable features in a worm-detection system: it requires deployment at only one place on the network (the gateway), it is capable of detecting many different types of worms, it does not rely on payload inspection, and it has a low false-positive rate. These features compare favorably with existing systems such as [9–11]. Solutions that depend on using worm signatures to identify the byte patterns sent from worm infected hosts may have difficulty in detecting polymorphic worms or worms which encrypt their payload during propagation, not to mention the cost of inspecting the payload of every packet in transit. Honeypot-based solutions typically capture connections to unused network addresses so will not detect a worm that only attempts to contact network addresses where a valid host exists. They are also vulnerable to spoofing attacks which deliberately send legitimate-looking traffic

to honey-pots causing real traffic to be flagged as worm traffic.

However, it is still unknown how fast and accurate we can be in detecting the infection of individual hosts. While SWORD provides an elegant solution for detecting the occurrence of zero-day worms, it is often necessary to know which hosts are infected in order to take prompt, concrete actions. In this paper, we enhance SWORD to further detect worm infections at the host level without requiring intrusive monitoring systems installed on end-hosts. Furthermore, we will outline a new methodology for evaluating the performance of a worm detection system using our Gateway Level Oregon Worm Simulator (GLOWS), and present a thorough evaluation of the speed and accuracy of the enhanced SWORD. We evaluate the effectiveness at identifying which hosts within a network are infected against a wide variety of worm propagation models, including random scanning, local-preference random scanning, and topological scanning. We also consider the effects of polymorphism and encryption of the worm payload.

The paper is organized as following. We begin by reviewing some previous work related to our research in Section 2, followed by our design for enhancing SWORD to detect worm-infected hosts in Section 3. In Section 4, we then describe our methodology for evaluating the speed and accuracy of the enhanced SWORD. Section 5 discusses our Worm simulator. Section 6 presents our results and the analysis of our experiments followed by Section 7 which analyzes the robustness of the alert threshold. More discussion is presented in Section 8, followed by our conclusions in Section 9.

2. Related Work

Recently there have been studies on how worms may behave [2, 6, 12–14] and the general requirements for containing them [7]. However, worm defense still remains largely an open topic. Research on worm defense typically falls into three categories.

- Intrusion detection systems (IDS) that identify suspicious behavior as it happens [11, 15–17].

- Rate-limiting suspicious outbound connections [18, 19].

- Performing forensic analysis of worms [3, 20, 21].

Our approach is to study and evaluate the detection of infected hosts in real time, so research from the first category is closest to ours. Research from the second category is not aimed at detecting which hosts are infected by worms. Research from the third category is complementary to real-time worm detection approaches such as ours but is generally done *after* the fact. In the following, we focus on the first category.

Real-time worm detection generally can be divided into two different categories: host-based IDS and network-based IDS. Whereas conventionally a host-based IDS detects whether or not a host is under attack and a network-based IDS detects whether or not a network is under attack [22], our work studies how well a *network*-based approach performs in detecting *host*-level worm infections.

Worm IDS can also be divided into misuse-based detection or anomaly-based detection. In detecting host infections of zero-day worms, one popular approach is to quickly discover a byte pattern of a zero-day worm and use that as the signature for detection [9, 10]; another is to set up a honeypot (which should not receive any traffic and thus any traffic it receives is probably malicious) and send out worm alerts upon the receipt of unexpected traffic [11, 23, 24]. The byte pattern approach can be used to detect if an individual host is infected or not, but because it needs to check the payload of the traffic, not only will it have a high amount of overhead, but it will also have difficulty in detecting polymorphic worms when the payload changes. The honeypot approach cannot directly help detect which local hosts are infected by worms, as a honeypot can only tell for sure that itself is being attacked. Additionally, honeypots suffer from the fact that they can only detect a worm if it scans addresses that are not populated by a regular host. That is, a honeypot will not ever detect a worm that only scans addresses with valid hosts.

As our research is on evaluating the speed and accuracy of the enhanced SWORD in detecting worm-infected hosts, we note that there are several other recent host infection detection systems geared towards worm detection.

- EarlyBird [9] is a signature generation system which has been shown to be effective in discovering zero-day worms. However, the system suffers from having to make a trade-off of false positives for speed. Furthermore, the system is not able to detect polymorphic worms.

- Moonwalk [25] is a system which determines the host which originated a worm attack. It is more of a forensic system, more useful after an infection has been discovered than in detecting an attack. Furthermore, it requires complete connection information – not just a trace at a network's gateway – limiting its overall utility.

- The early warning system from Zou et al. [26] is also able to quickly detect Internet worms. The focus of their work, however, is to detect the presence of worms in the Internet at large, not to detect which hosts are infected.

- Snort [27] is probably the most widely used IDS, and it is entirely signature based. In order for it to be useful against zero-day worms, another system must be used to supply Snort with the proper signatures.

3. Enhancing SWORD

In order to evaluate the performance of detecting the worm infection at host level we must enhance SWORD so that it is able to detect whether or not a particular host is infected with a zero-day worm. In this section, we first describe how SWORD works, then describe our enhancements.

3.1 SWORD: Self-propagating Worm Observation and Rapid Detection

The SWORD system detects worms by monitoring connection activity and watching for patterns of traffic that are expressions of some of the essential characteristics of worm behavior. The monitor itself sits at a network's gateway, a vantage point from which all traffic into and out of the network is visible. This is a scheme that is minimally intrusive from a deployment standpoint, requiring far less administration than solutions that require installations on each host of the network, but as we will show it is still an extremely effective vantage for detecting worm infections. It does not allow for the observation of internal network traffic, meaning that a purely local worm would not be detected; but in order to spread beyond the local network a worm must make connections that cross the gateway, so we do not view this as a significant limitation.

The basic unit that SWORD works with is a *connection* between two hosts. This connection can utilize either of the standard Internet protocols: being a TCP (Transmission Control Protocol) connection from set-up to teardown, or a series of UDP (User Datagram Protocol) packets between two hosts and sharing a common destination port. We track information such as the source and destination ports, and the TCP flags sent, but SWORD does not examine the connection payload, so the resources required for assembling and tracking connections are small. After assembling the information about a particular connection, it is run through a pair of heuristics to determine the likelihood that this connection is a worm connection. The heuristics look for individual characteristics of worm behavior.

The *Causal Similarity Heuristic* leverages the fact that attacks against a given vulnerability must share some similarity in connection attributes. Connections are considered wormlike when a series of similar connections can be found in a causal connection graph. We explicitly avoid examining the payload itself for similarity since polymorphic or encrypted worms may vary their payload greatly. Instead, the heuristic considers connection attributes such as protocol, target port, and TCP flags that have been set. Once payload similarity is removed from consideration, however, the sensitivity of the similarity comparison is limited. By reducing the scope of the connections we look for similarity in to only those relevant to worm propagation (those connections that could have caused this connection) we reduce the false positives to a manageable

level. SWORD maintains a *Causal Connection Graph* where each node represents a connection and each edge represents potential causation. With the limited information available to the monitor, this graph is not expected to show the entire infection path of the worm, but it does allow us to limit the connections we compare against when looking for similarity.

The *Destination Address Distribution Heuristic* examines the pattern of destination addresses for a given host. A normal host shows a distinctly Zipf-like pattern when the destinations are ranked by popularity and the log of the rank of each destination is plotted against the log of the number of visits to that destination. When infected with a worm however, the pattern changes. This heuristic works by watching the trend in the shape of the destination address distribution and considers a connection to be wormlike after a sufficient number of connections have trended the pattern away from Zipfness.

The individual heuristics suffer from false positive rates that are too great to allow us to rely on their output directly. Instead, SWORD only considers a connection to be wormlike when both heuristics agree, and then monitors the level of wormlike connections over a sliding window. When a threshold of allowable wormlike connections is exceeded, SWORD raises the alert that a worm is active within the network. SWORD establishes the thresholds during a training period and maintains separate sliding windows and thresholds for TCP- and UDP-based worms.

Further details regarding the inner-workings of SWORD can be found in our technical report [8].

3.2 Enhancing SWORD to Detect Worm-Infected Hosts

Whereas SWORD is able to detect whether a domain has a zero-day worm, we enhance SWORD so that we can also detect which individual hosts are infected. We do so without requiring intrusive monitoring systems installed on end-hosts, but instead follow the same spirit of the original SWORD system. If the number of wormlike connections emanating from a particular host during a given window exceeds the host-level threshold, then we consider that host to be infected.

The process of obtaining the host-level alert thresholds is similar to the process employed by the original SWORD for domain-level thresholds. By observing normal traffic from individual hosts during a training period, one can obtain the total number of connections that the two heuristics would label as wormlike. The maximum observed wormlike connections within a window for any host in the domain, or some multiple of it, can be used as the host-level alert threshold for detecting a worm infection.

Such enhancement is straightforward, but it serves two important purposes: (1) as in the original SWORD, it continues to be a gateway-based approach to detecting worms and is transparent to individual hosts; and (2) the per-

formance of host infection detection can then be evaluated based on an approach that we believe is superior to payload-inspection-based or honeypot-based worm detection approaches.

The knowledge of which hosts are infected can be used to perform forensic analysis of a worm outbreak or to intelligently quarantine individual hosts, but evaluating such countermeasures is beyond the scope of this work.

4. Methodology

We adopt a trace-based simulation approach to evaluating the enhanced SWORD in detecting host infection. In the following, we first describe what metrics we use for evaluation, and then describe the traffic we choose in the evaluation.

4.1 Metrics

The metrics we use must be able to evaluate the following: the *latency*, or the average speed with which that any given host is detected to be infected with a worm, and the *accuracy* with which any infected host is identified.

The latency is simply the time it takes to detect that an infected host is infected, which we calculate by subtracting a host's time of infection from its time of detection. We report the value averaged across all of the hosts infected for a given run of the experiment

The accuracy is slightly more complex. At a basic level we need to know *false positives* (how many non-infected hosts were flagged as being infected by the system) and *false negatives* (how many infected systems were *not* flagged as being infected). Moreover, we will need to know *adjusted false negatives* to find out how accurate we are in detecting those hosts that initiate more worm connections than the level defined by the threshold. This measure is specific to our experimental setup and filters out those false negatives occurring from a host being infected immediately before the termination of the experiment. We feel that this adjusted measure more accurately represents the true performance of the enhanced SWORD because it does not penalize SWORD for the limitations of our experimental setup. We will use the false negative and adjusted false negative statistics to present *accuracy* and *adjusted accuracy* percentages which are, respectively, the percentage of worm infected hosts correctly detected and the percentage of worm infected hosts correctly detected which sent more than the threshold amount of worm connections (false positives are not addressed as they did not occur in our results).

4.2 Background Traffic

We use a pre-recorded network trace from a real network as the background traffic in our experiment. We did not

Table 1. Auckland-IV trace details

Date	Active hosts	Out Conns.	In Conns
2001/03/06	2344	2,459,281	979,366
2001/03/07	2270	2,352,294	929,511
2001/03/08	2296	2,263,636	1,074,695
2001/03/09	2283	2,328,105	864,532

use a live network feed because we need to run controlled and repeatable experiments. Simulated traffic was not an option because it there simply is no way to simulate traffic with the realism that we require.

The real trace in our experiments is the Auckland-IV trace [28]. It is a continuous 45 days GPS-synchronized IP header trace recorded between February and April 2001 at the University of Auckland and Auckland University of Technology. Traffic was tapped from an OC3 ATM link that connects the universities to the service provider. The inside networks contain two /16 and several /24 prefixes and all IP addresses are anonymized in the trace. The trace includes all the TCP and UDP header information necessary for our experiments, but no payload information. We redistributed the anonymized IP addresses from the trace into a fictionalized IP range that properly reflects the topology of the network inside the Auckland border router. On any given day there were approximately 2250 hosts making roughly 2,300,000 total outbound connections and another 960,000 incoming connections (see Table 1). The hosts which were active varied from day to day, and roughly 5000 total internal hosts were active at some point in the trace.

We do not know, of course, whether there are any worms active in this trace, or which connections in the trace represent those initiated by an infected host, so there is no way to test the effectiveness of the enhanced SWORD against this trace alone. Instead, we make the assumption that this trace has no worm traffic in it whatsoever, and then we inject our own simulated worm traffic, which we discuss in the following section.

4.3 Worm Traffic

We have created our own worm simulator, GLOWS (see Section 5), to model the spread of the different types of worms used in our experiments. It uses a network topology that matches that of the Auckland trace – two internal /16 networks separated from the Internet by a border router (see Table 2 for details.), and captures traffic that crosses the border router to a trace file. We then use this worm trace as an input to our evaluation of the enhanced SWORD.

In order to test our detectors effectiveness across a broad range of scenarios, we simulated the following worm scanning models: random, permutation, partition, sequential, local preference, and topological. Our initial

Table 2. Network topology details

External network details	
Total addresses	$\sim$ 4 billion (2^{32})
Hosts	$\sim$ 300 million (from isc.com)
Service runners	3 million
Vulnerable hosts	300,000
Internal network details	
Total addresses	$\sim$128,000 (2^{17})
Hosts	5000 (from trace)
Service runners	500
Vulnerable hosts	500

analysis found that our detector performed nearly identically for the random, permutation, partition, and sequential scanning worms, so we omit results from all but the random scanning worm. Additionally, GLOWS supports polymorphic payloads for any of the propagation models, but we do not include these simulations because SWORD does not examine payload characteristics.

In addition to the scanning algorithm used, parameters for the target port number, payload size, and network protocol must be selected. All our simulations use port 80 as the target port. This is one of the most numerous ports used in the Auckland trace (roughly 50% of connections go to port 80) making detection as hard as possible as the wormlike connections look more like legitimate traffic than if they were on another port. The size of the packets transmitted is not considered by the detection heuristics and thus is irrelevant. Finally, all of our simulations are run for both TCP and UDP traffic.

To create the worm traces used in our experiments, we begin with 3000 hosts infected in the Internet and a worm connection crossing the border router and infecting one of our internal hosts. This is meant to replicate a reasonable scenario for a zero-day worm in the early stage of its development while also reducing the time required to run our simulations. We run each simulation until 100,000 worm connections have crossed the gateway, or 1 h has passed in the simulation, whichever occurs first. This limits our simulation to the early phase of a worm's propagation when congestion and patching effects are likely to be minimal. Every scenario is run 10 times to create 10 unique worm traces for each speed and propagation combination.

4.4 Merged Traffic

To run experiments against the enhanced SWORD, we create a merged trace to be the input. The merged trace consists of a single day's worth of connections just from the Auckland trace to warm up the heuristics, followed by connections interlaced from the desired worm trace and the next day's Auckland trace. These connections are interleaved based on the connection start-time, resulting in a single trace where we can identify which connections are worm connections and which hosts are infected at what time, allowing us to accurately measure the performance of our detection system.

5. GLOWS: Gateway Level Oregon Worm Simulator

Our worm simulator, GLOWS (Gateway Level Oregon Worm Simulator), simulates the spread of a worm across the Internet and its propagation into a single domain; with the goal of capturing the worm traffic that crosses the gateway point separating the monitored domain from the Internet.

5.1 How GLOWS Works

GLOWS uses a finite-state model to simulate the behavior of each vulnerable host in both the Internet and the internal network. This provides a greater degree of accuracy than probabilistic simulators, particularly for worms such as the topological scanning worm which must know of other hosts running the service. Modeling each vulnerable host is feasible for a number of reasons including: the number of such hosts is only a small fraction of the total hosts in the Internet, we model host interactions at the connection level rather than the packet level, and because we do not model congestion effects or background traffic within the simulator. GLOWS loses a small level of accuracy due to ignoring the effects of congestion, but if experiments are focused at the beginning of the infection cycle where relatively few hosts are infected, this impact is minimal.

To support studying a broad variety of worm scenarios, GLOWS supports the following parameters.

- *Connection type.* Connections can be either TCP-based or UDP-based.

- *Scan speed.* The speed at which the worm scans can be varied.

- *Scan type.* Using previously presented worm studies by Staniford et al. [2], we were able to model the following scanning types: random-scanning, sequential-scanning, permutation-scanning, partition-scanning, local-preference-scanning, and topological-scanning.

The various scanning techniques work as follows: *random-scanning* worms are the simplest as they simply choose each new target address completely randomly. *Sequential-scanning* worms choose a random address to start from and scan sequentially from there. *Permutation-scanning* worms permute the entire address space and sequentially scan that. *Partition-scanning* worms partition

the address space and scan only their partition. *Local-preference* worms choose target addresses randomly, but with a preference towards choosing from the local subnet: they choose an address from their host's class B address space with a 50% probability, from its class A address space with a 25% probability, and from the Internet as a whole with a 25% probability. Finally, the *topological-scanning* worm starts with a list of roughly 500 addresses known to be running the target service, although not necessarily vulnerable. Once these have been contacted, it reverts to a pure random-scanning worm.

GLOWS can model the network topology of the monitored domain based on host information taken from a network trace, allowing it to create realistic worm traffic for a given monitored domain. Addresses where a host is active in the internal network are derived from detected host activity in the real trace, and external hosts are probabilistically allocated. Not all hosts run the service that the worm is attacking; we assign service runners probabilistically in both the internal and external network.

GLOWS accurately models the connection-level interaction between two hosts during an infection attempt, right down to setting appropriate TCP flags. Scanning attempts to non-existent hosts result in SYN/RST exchanges as one would expect to see in the real world, and the worm client can disconnect at any point leaving the connection in an arbitrary state. The payload and target port number are also configurable.

5.2 Implementation Details

GLOWS is implemented as a Java program comprised of 20 classes and approximately 4000 lines of code. The internal network and the external network (i.e., the rest of the Internet) are implemented in separate classes allowing the internal network to be modeled more accurately and the external network to use a less accurate but higher speed implementation. There are four possible states for every IP address: no host, host, host running service, and host running vulnerable service.

For the addresses in the internal network, we load the list of hosts from a configuration file produced by analysis of the real network trace that the internal network is modeled after. Service runners and vulnerable hosts are chosen randomly from within this set. Every host in the internal network is an instance of our ActiveHost class, allowing each host to respond independently to connection requests.

For the external network, the sets of vulnerable addresses and service runner addresses are generated by a random number generator and are stored as simple lists of 32-bit integers to conserve memory. A given address in the Internet is determined to have a host or not by the following procedure. First check the list of vulnerable addresses, then the list of service runner addresses; if it is present in neither of these lists, it may still have an active

host. Whether or not this address has a host is randomly determined, but must be consistent throughout a given simulation: a random number is generated after seeding the random number generator with the address in question plus a simulation specific salt value. If the random number is below the configured value for the percentage of hosts in the external network, then this address has an active host. This process ensures that throughout a simulation the set of addresses with hosts is consistent, but for multiple simulations we will get different sets of addresses with hosts. It can be performed efficiently through the use of a Mersenne Twister [29] implementation as our random number generator, which has excellent performance and very low overhead for setting the seed.

Interactions with hosts in the external network are resolved by the state of the address except that when a vulnerable address is infected an ActiveHost instance is created for that address to respond to future interactions.

At each time tick, the ActiveHost instances in both the internal and external networks are polled for activity. Those that are infected and actively sending worm scans will generate their next scan based on the worm implementation currently active. Because each ActiveHost has its own instance of the worm, individual hosts can have unique neighbor lists, network partitions, or other host-specific information. The worm implementation is pluggable, allowing us to simulate a number of different scanning and propagation schemes.

At the end of the time tick all of the worm scans are processed. There is a complex set of possible results for each connection attempt, as the worm implementation can specify all of the connection attributes and TCP flags for each kind of connection: to a dark address, to a non-service running host, to a service running host, to a vulnerable service running host, and to an already infected host. Connections from internal to internal hosts or external to external hosts are first processed, and then those connections that cross the gateway are processed. The complete set of attributes for connections that cross the gateway are logged to a file for use as input in the worm simulator experiments.

5.3 Simulator Performance

The speed with which a complete simulation can be processed is, of course, directly correlated with the number of vulnerable hosts and the rate at which hosts make connections. The following performance numbers are measured when running simulations on a 1.73 GHz Pentium M Laptop with 512 MB of RAM.

A basic scenario for our experiments is the high-speed (100 connections per second) random scanning worm with a vulnerable population of 300,000 hosts, of which 3000 start out as infected. We run this simulation until 100,000 worm connections have crossed the gateway (roughly 160 s of simulated time), and it takes approximately 24 min to complete.
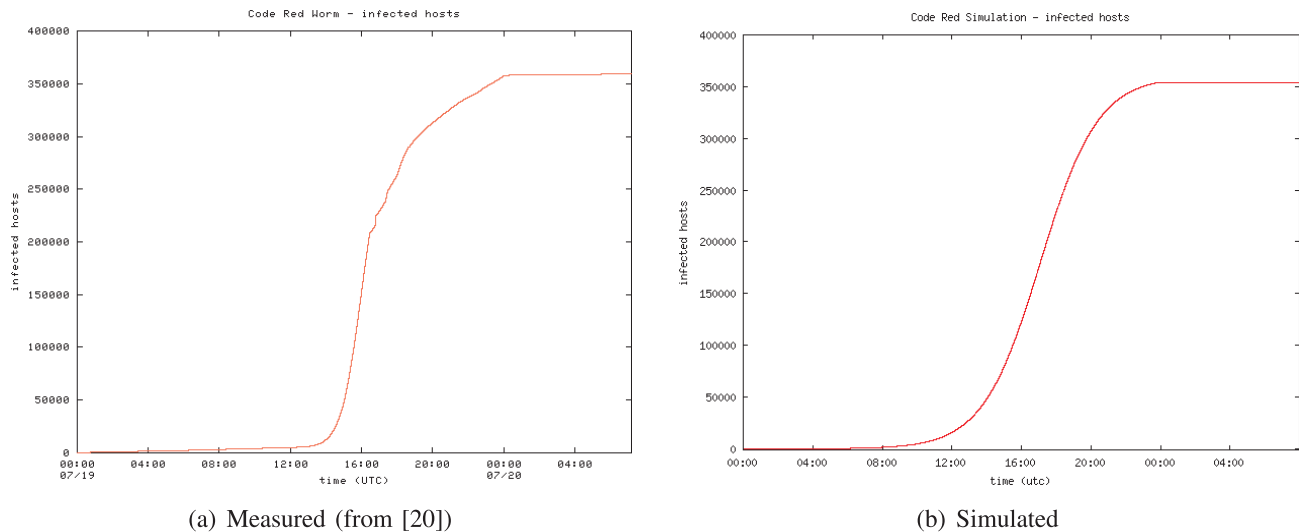
(a) Measured (from [20])

(b) Simulated

Figure 1. A comparison of the spread of the measured and simulated Code Red worms showing the cumulative count of IP infected addresses

This contrasts with the same scenario but run for a low-speed (1 connection/s) worm where our simulation terminates after 3600 s of simulated time, and only 4500 connections crossing the gateway, but which runs in just over 1 min.

Larger simulations take correspondingly longer to run. Our simulation of the Code Red v2 worm on the same hardware took 56 h when run until 99% of the vulnerable population of 360,000 hosts were infected. This simulation generated more than 12,901,600 connections which crossed the gateway (and of course substantially more that didn't) over more than 24 h of simulated time.

5.4 Simulator Validation

Before we can analyze the success or failure of our system in detecting worms, we first need to ensure that our experiments are realistic. We know that the legitimate traffic is realistic because it is real, captured traffic, but we must validate that GLOWS produces a realistic looking infection trace. The easiest way to do that is to attempt to simulate an actual worm and compare simulation results with the real world results.

In Figures 1a and b we compare the spread of a simulated version of the Code Red worm with its spread during the real world outbreak as measured by Moore et al. in [20]. The simulation was run with propagation and vulnerable population parameters similar to the real Code Red v2 Worm (see Table 3). In both curves we see a similar basic shape with the inflection point occurring around 16 h after the start of the outbreak. The curves begin to diverge after hour 18, most likely due to the impacts of both patching infected systems and network wide congestion. Our experiments are focused around the very early outbreak of

Table 3. Code red simulation parameters

Parameter	Values
Vulnerable hosts	360,000
Initial infected hosts	3
Connection rate	2 conns/s
Scanning strategy	Random scan

the worm, however, where our simulation appears to be quite accurate.

6. Results

In this section, we first present our results in selecting thresholds for detecting host infection, then report the accuracy and latency in detecting the host infection.

6.1 Window-size and Threshold Selection

The *sliding window size* and the *threshold* are two key factors in the success of the enhanced SWORD system. The sliding window size determines the length of the period over which the wormlike connections from a given host are counted. The threshold differentiates the number of wormlike connections from a host's normal traffic – also called *background noise* – and that from a worm-infected host. The heuristics we utilize are expected to generate low levels of false positives at the connection level, perhaps labeling certain legitimate connections from a host as wormlike. It is important that the threshold is set in such a way that these false positives, or background noise, do not trigger an alert that a host is infected.
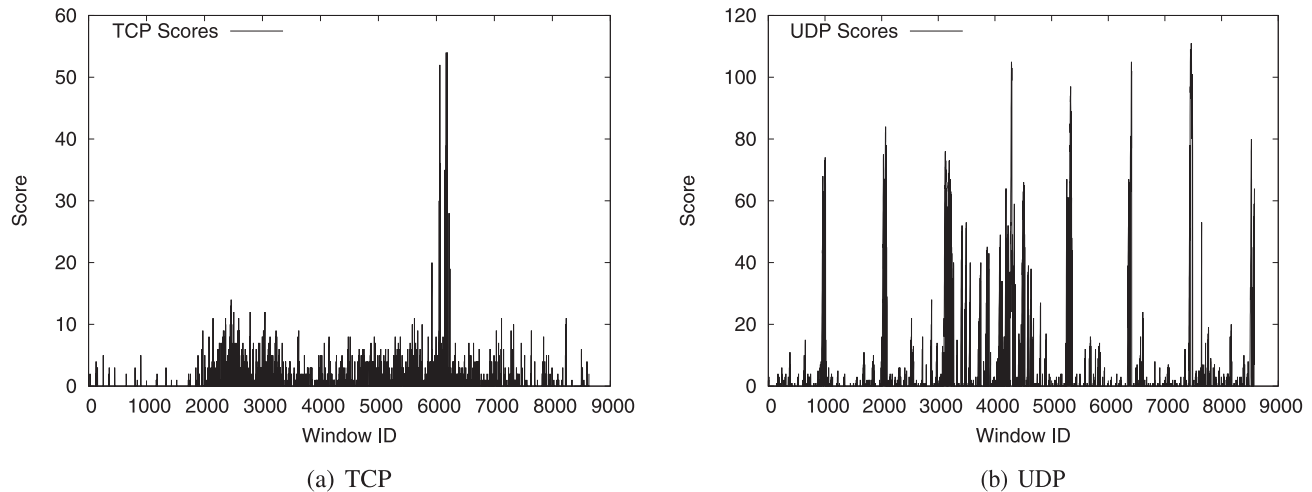
Figure 2. Background trace scores vs. time. Each score is the number of wormlike connections from a single host during the sliding window ending at this time

Table 4. Background trace score details

(a) Background trace TCP score details		
Date	Mean window score	Max. window score
2001/03/06	2.88 ± 2.29	20
2001/03/07	2.94 ± 2.26	12
2001/03/08	3.27 ± 4.40	54
2001/03/09	2.66 ± 2.02	13
Window: 2 min, Threshold chosen: 55		
Date	Mean window score	Max. window score
2001/03/06	22.23 ± 22.05	109
2001/03/07	24.66 ± 24.78	95
2001/03/08	31.03 ± 27.61	111
2001/03/09	28.23 ± 27.54	99
Window: 4 min, Threshold chosen: 112		

The selection of the sliding window size and the threshold is a compromise between detection sensitivity and latency. Our goal is to choose the smallest window size that still yields enough sensitivity to detect even slow-moving worms. Evaluation of the background (non-worm) traces reveals that the connection-level false positives tend to be quite bursty, as can be easily seen in Figure 2a and b and Table 4a and b.

As we increase the size of the sliding window, it increases the absolute value of the threshold, but reduces the average number of connections per unit of time needed to exceed the threshold. A larger window is therefore desirable as it allows us to detect worms with slower scanning rates. The size of the window also impacts the speed with which we can detect worms, however. We must identify

enough worm connections from a given host to exceed our chosen threshold, so as the threshold grows with the window size, the number of connections a worm must send to exceed it also grows. A smaller window is therefore desirable to maintain low detection latencies.

We chose our TCP and UDP thresholds and window sizes based on a training period for our experimental domain. For this evaluation we used the first 3 days of the Auckland trace as our training period then performed our experiments on the final day using the threshold established. To obtain the threshold we ran the enhanced SWORD against the (assumed to be non-worm) connections from the Auckland trace and measured the average and maximum wormlike connection counts for various window sizes. We selected a 2-minute window size for TCP connections with a threshold of 55 wormlike connections, and a 4-min window for UDP connections with a threshold of 112 wormlike connections (Table 4a and b). These thresholds are higher than any measured bursts of false positives in the background traces, and substantially higher than the average observed value.

In researching the source of the background noise in the traces, we discovered that many of UDP false positives were triggered by DNS activities. This is not surprising, as DNS lookups may generate many similar connections to many different hosts, matching the criteria of both of our heuristics. This implies that white-listing the DNS connections would improve our results by lowering our UDP window size and threshold (see Table 7 below). However, this would also create a hole in our coverage that could be exploited by a worm targeting a DNS vulnerability. In the remaining results sections, we will present our results *without* DNS white-listing save for a brief discussion in Section 6.4.

Table 5. Accuracy: high speed worms

(a) Accuracy: high speed TCP worms		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	98.49 ± 2.30	100 ± 0.0
Local preference	88.08 ± 2.26	95.69 ± 1.30
(b) Accuracy: high speed UDP worms		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	97.22 ± 2.91	98.72 ± 2.65
Local preference	79.55 ± 2.42	96.92 ± 0.75

Table 6. Accuracy: low speed worms

(a) Accuracy: low speed TCP worms		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	99.17 ± 2.50	100 ± 0.0
Local preference	80.56 ± 1.84	87.54 ± 1.89
(b) Accuracy: low speed UDP worms		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	96.81 ± 4.97	100 ± 0.0
Local preference	77.210 ± 2.00	95.01 ± 1.34

6.2 Detection Accuracy

Having established a suitable threshold, we can now examine the accuracy of the enhanced SWORD at identifying worm infected hosts.

We begin by evaluating the enhanced SWORD's performance against high speed worms. The results for both the TCP and UDP experiments are presented in Table 5a and b and show the enhanced SWORD's exceptional performance.

The enhanced SWORD correctly identified 100% of the infected hosts with zero false positives in all of our experiments for the TCP-based random scanning and topological scanning worms (Table 5a). These worms are more likely to initiate a worm connection to an address across the gateway than the local preference worm is, so they are easier for our system to detect. The local preference worm sends fewer connections across the gateway, which allows for normal traffic from the infected host to have a better chance of interfering with our detection mechanisms. Even so, we correctly identified more than 95% of the infected hosts in the local preference experiments.

The performance against UDP worms is not quite as good (Table 5b), due to the increased threshold and window size. Each infected host must produce more identified worm connections to exceed the threshold, and the longer window allows more time for normal traffic to interfere with correct connection categorization. Even so, we correctly identify 100% of the infected hosts for the random propagation model and more than 96% of the infected hosts for the topological and local preference worms.

The enhanced SWORD's performance against low-speed worms is similar, but suffers somewhat at detecting TCP local preference worms (Table 6a and b). One might expect that detecting the UDP local preference worm would be more difficult than the TCP-based version due to the higher UDP threshold, but instead it seems to be the shorter TCP window that limits detection efficacy.

6.3 Detection Latency

We have seen that the enhanced SWORD can accurately identify worm-infected hosts, but the question remains: can this be done in a timely fashion? The Sapphire/Slammer worm was able to infect most of its vulnerable population in under 10 min [3], setting a benchmark we must be able to beat to field an effective worm detector. In fact, our results show that we are able to detect worm activity substantially faster than 10 min for both high-speed and low-speed worms.

When run with our high-speed worm variants, the enhanced SWORD was able to detect infected hosts in under 3 s in all cases (see Figure 3a). This shows that the enhanced SWORD is indeed capable of countering a worm such as Sapphire/Slammer. Note also that the connection rate of this high-speed worm is reasonable in comparison to the Sapphire/Slammer. The Sapphire/Slammer worm is bandwidth limited, and a host on a 100 Mbit/s connection would be capable of sending approximately 300 connections/s. This is greater than the 100 connections/s of our experiment and would therefore be even easier for the enhanced SWORD to detect.

There is little variation in the high-speed results between the different worm types because these high-speed worms produce enough wormlike connections to overwhelm our thresholds.

There is substantially more variation between detection latencies of the various worm propagation models in the low-speed experiments (Figure 3b). The enhanced SWORD is substantially slower a detecting low speed worms due to the lower volume of worm connections, but even so it detects all worm varieties in under 5 min, and all but the Local Preference worm in under 2.5 min. These times are again faster than the benchmark set by the Sapphire/Slammer worm, and in any case slow scanning worms will be much slower at infecting the entire Internet.

The average latency for detecting slow speed worms which use the local preference scanning method is substantially higher than that of worms which use other scanning methods. This is a direct result of the fact that only
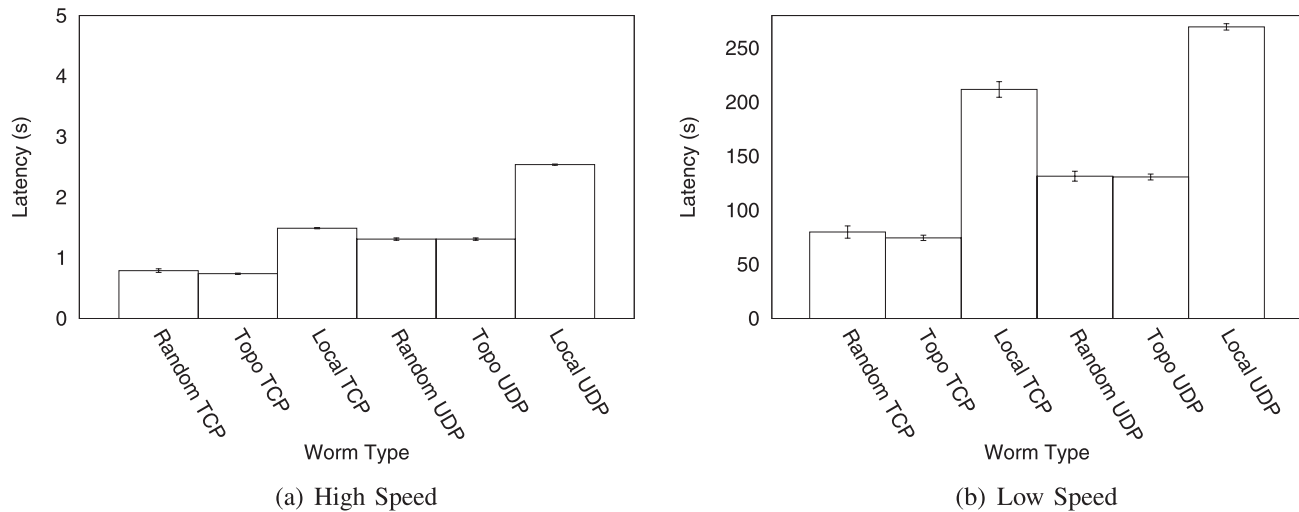
(a) High Speed	(b) Low Speed

Figure 3. Average detection latency for worms

Table 7. Background trace UDP score details with DNS white-list

(a) Background trace TCP score details		
Date	Mean window score	Max. window score
2001/03/06	4.38 ± 6.61	49
2001/03/07	4.31 ± 6.11	49
2001/03/08	2.44 ± 1.87	16
2001/03/09	3.53 ± 3.51	36
Window: 2 min, Threshold chosen: 50		

Table 8. Accuracy: UDP worms with DNS white-list

(a) Accuracy: high speed UDP worms with DNS white-list		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	98.50 ± 2.30	100 ± 0.0
Local preference	90.00 ± 2.03	96.72 ± 1.06
(b) Accuracy: low speed UDP worms with DNS white-list		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	99.17 ± 2.50	100 ± 0.0
Local preference	89.900 ± 1.52	96.69 ± 1.16

50% of the connections from the local preference worm pass through the gateway and the monitor. These slow speed worms only make one connection per second, so every connection that does not pass through the monitor slows detection by roughly a second.

There is a clear difference in the TCP latencies versus UDP latencies because of the different windows and thresholds used, as described in Section 6.1.

6.4 DNS White Listing

The high threshold and long window size used for UDP worms negatively impacts the enhanced SWORD's detection latency. The threshold and window size choices were influenced substantially by false positives stemming from DNS traffic (as was mentioned in Section 6.1), but if DNS traffic were exempted from consideration by the enhanced SWORD, we could reduce both the window size and the threshold which would allow for superior performance (Table 7).

The data in Table 8a show that our detection accuracy actually increases marginally for high-speed worms when we employ the DNS white-list, with the topological worm detection rate increasing from 98.72 to 100%. Low-speed worms see a similar improvement (see Table 8b) although in no case is there a dramatic improvement.

More substantial improvements can be seen in the latency results, particularly for low-speed worms (see Figure 4b). The average detection latency for random and topological scanning worms was reduced by half from roughly 130 s to less than 70 s. The local preference worm saw a similar improvement being reduced from 270 s to less than 130 s.

The improvement for high-speed worms (Figure 4a) is equally impressive, with the local preference results improving from 2.5 to 1.3 s and the random and topological worms improving from 1.3 to 0.7 s.

Clearly, exempting DNS traffic from our monitor improves performance. Reducing the detection latency of slowly scanning worms reduces the damage done before they are detected and could be extremely important in lim-
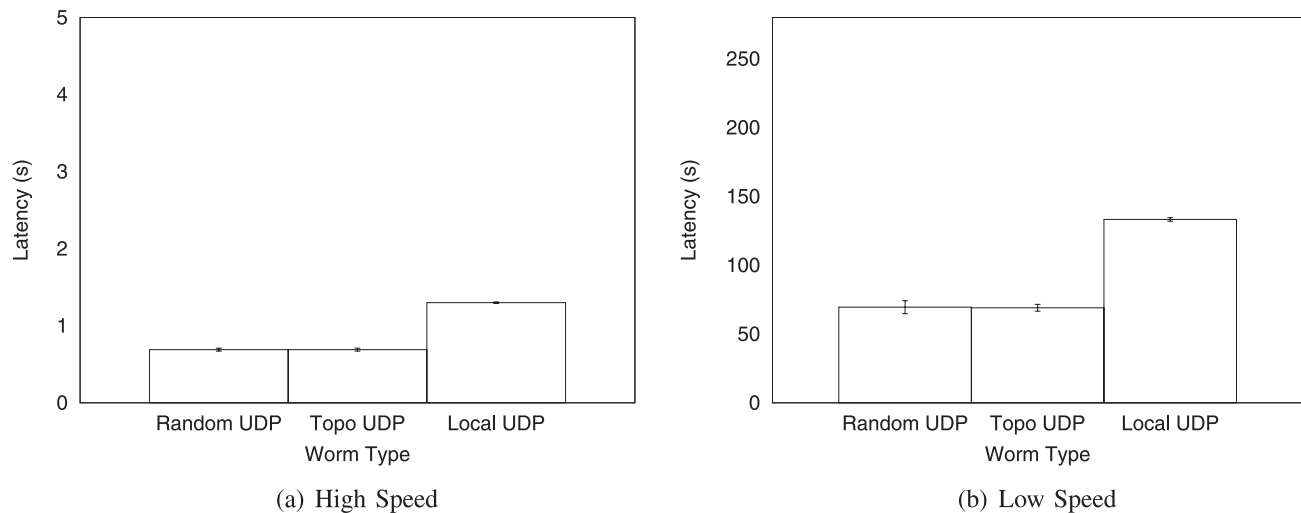
Figure 4. Average detection latency for worms with DNS white-list

iting the overall impact of a worm outbreak. On the other hand, failing to monitor DNS traffic would leave the network wide open to a worm exploiting a DNS vulnerability.

7. Threshold Robustness

For the main body of our results we chose an aggressive threshold with no margin of safety above the maximum background noise observed during training. This balances the desire to limit the false positives with that of providing optimal detection. However, the threshold may be varied substantially, while still providing effective detection. One could lower the threshold *below* the maximum observed noise, almost assuredly leading to false positives, but also resulting in a very sensitive detector that would detect worms quickly. On the other hand, one could increase the threshold substantially *above* the maximum observed noise to further reduce the chance of false positives. This decreases the sensitivity of the detector to slow-moving worms and increases the detection latency, but the trade-off against future false positives may be worth it. Here we present our results when doubling both the TCP and UDP thresholds, while maintaining the same window size for each.

7.1 Detection Accuracy

For high-speed worms, the results in Table 9a and b show that detection accuracy decreases marginally but with an adjusted accuracy still above 95% for all worm types. This represents a decrease in accuracy of less than 1% in almost all cases. The high speed worms produce connections at a sufficient rate that there is quite a bit of room for increasing the threshold.

Table 9. Accuracy: high speed worms with 2× threshold

(a) Accuracy: high speed TCP worms with 2× threshold		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	97.22 ± 2.91	98.72 ± 2.65
Local preference	78.91 ± 2.41	95.84 ± 0.87
(b) Accuracy: high speed UDP worms with 2× threshold		
Type	Accuracy	Adjusted accuracy
Random	99.38 ± 1.88	100 ± 0.0
Topological	94.88 ± 4.76	98.99 ± 2.03
Local preference	61.47 ± 2.86	96.08 ± 2.03

The results for the low-speed worms presented in Table 10a and b show a slightly different story. For these worms the adjusted detection accuracy is actually still perfect for the Random and Topological worms, but *none* of the Local Preference infected hosts were detected. The reason for this is that the Local Preference worm sends only roughly half of its connections outside the network and only those connections going out of the network are visible to the monitor. This lowers the connection-level detection accuracy for Local Preference worms because there are fewer connections to disrupt the destination address distribution and fewer connections to match in the similarity comparison. This means that for a given unit of time, the monitor will observe fewer than half the number of wormlike connections for a host infected with the Local Preference worm than it will for a host infected with another type of worm. In this case, the reduced number of observed wormlike connections is less than the now increased threshold, and no infected hosts are detected. See Section 7.3 for further discussion of this point.

(a) High Speed
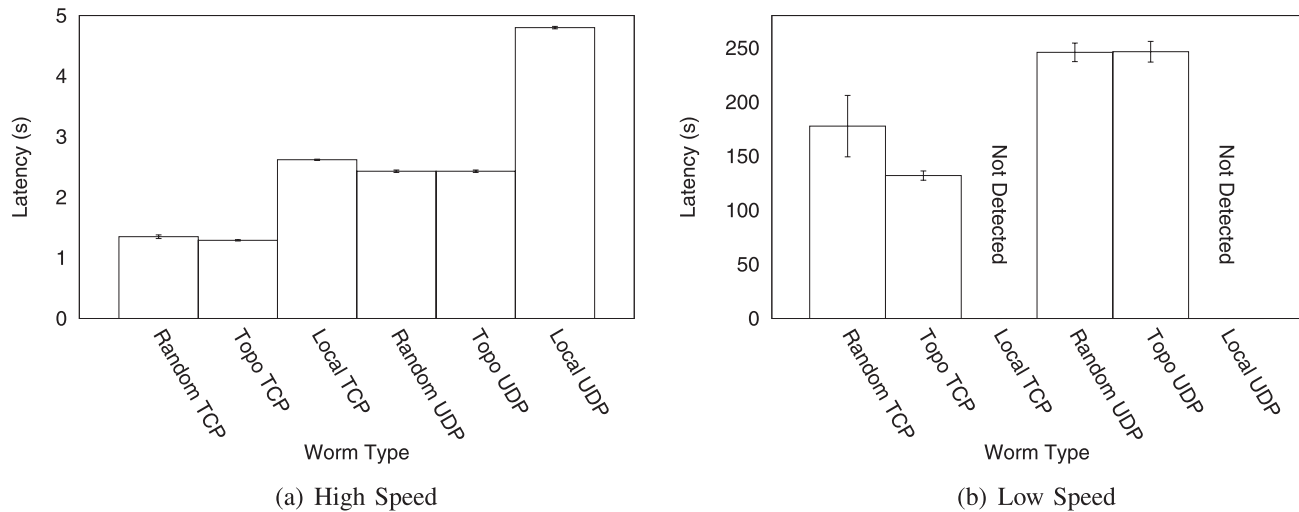
(b) Low Speed

Figure 5. Average detection latency for worms with 2× threshold

Table 10. Accuracy: low speed worms with 2× threshold

(a) Accuracy: low speed TCP worms with 2× threshold		
Type	Accuracy	Adjusted accuracy
Random	100 ± 0.0	100 ± 0.0
Topological	98.61 ± 4.97	100 ± 0.0
Local preference	0.00 ± 0.00	0.00 ± 0.00
(b) Accuracy: low speed UDP worms with 2× threshold		
Type	Accuracy	Adjusted accuracy
Random	97.50 ± 7.50	100 ± 0.0
Topological	95.69 ± 5.36	100 ± 0.0
Local preference	0.00 ± 0.00	0.00 ± 0.00

7.2 Detection Latency

The detection latencies with the doubled thresholds shown in Figure 5 are slightly less than double the detection latency with the original threshold. They are not precisely double because early worm connections are not recognized as wormlike because it takes a series of worm connections to disrupt the destination address distribution and have a sufficient number to show similarity. Early worm connections are often not recognized as wormlike, but later worm connections are identified as wormlike with high accuracy. For the high-speed random TCP worm, it was identified with an average latency of 0.79 s with the original threshold but only 1.35 s with the doubled threshold.

7.3 Threshold Analysis

We have seen that increasing the threshold may cause some types of slow-scanning worms to not be detected. We can derive the minimum rate at which a worm must generate connections to be detected by a given threshold. Worms that scan below such a rate would not be detected, but if the rate is low enough, a worm scanning that slowly would be limited in the damage it could cause and the rate at which it could spread.

The minimum scanning rate the enhanced SWORD will be able to detect for a given threshold τ and window size w can be derived as follows: assuming that we detect worm connections with some accuracy a, the infected host must generate worm scans with a rate r such that when combined with the false positives for the current window f_w, the following is true:

$$(r \times a \times w) + f_w > \tau. \tag{1}$$

The connection-level false positives f_w are dependent on the background traffic for a given window, but for evaluation we can consider the worst case where they are zero.

In our experiments with a random-scanning TCP worm, we detected worm connections with an accuracy of roughly 97% and used a threshold of 110 for a 120 s window. Plugging these numbers into the equation, we can see that we would detect a worm-infected host within our network as long as the worms scanning rate was greater than 0.94 connections/s.

Note that our accuracy at detecting the connections of a Local Preference scanning worm is substantially lower as the monitor has the opportunity to observe only half of the worms scans. Assuming that our accuracy at detecting Local Preference worm scans is 97% of the 50% of the connections the monitor sees we would be able to detect

an infected host if the worm scanned with a rate > 1.89 connections/s. This value is greater than the speed in our experiment which is why we failed to detect the low-speed worm.

During a window with higher connection-level false positives for legitimate traffic (and we know that they spiked as high as 111 within a single window for a UDP worm during the training period), a worm with an even slower speed would be detected. Perhaps had we run our low-speed experiments for longer than 1 h and encountered a window with high background noise, we would have detected the low-speed Local Preference worm.

8. Discussion

In addition to the above discussion regarding the results of our evaluation, there are some other topics which deserve attention.

8.1 Worm Speed

In our evaluation we considered worms which had both high speed (100 connections/s) and low speed (1 connection/s) propagation speeds. Of course there are other speeds that worm authors could use, including connection rates of even less than 1 connection/s. If a worm is slow enough that the traffic it generates is interspersed throughout a large amount of normal traffic, it becomes much more difficult to detect. It becomes more difficult for the Causal Similarity Heuristic to find similar connections, and there are too few worm connections to measurably affect the destination address distribution, thwarting the Destination Distribution Heuristic as well. On the bright side, if a worm propagates so slowly as to be undetectable it is also likely to be too slow to be a real danger.

8.2 Modern Traffic Traces

For our background traffic, we would have preferred a more modern trace including modern peer-to-peer (P2P) traffic, but were unable to find any which met all of our needs. Some readers may worry that P2P traffic would alter the destination address distribution of legitimate traffic, causing false positives or increasing the required detection threshold. However, we are confident that P2P traffic would not seriously affect our results. Consider, for example, Gnutella. Leaf Gnutella nodes usually connect to a small (around 3) number of peers directly and even the core, or ultrapeer, nodes generally only connect to around 30 peers. BitTorrent clients similarly receive only a short list of peers to contact, and DHT (distributed hash table) nodes generally contact only those few nodes in their neighbor lists. These numbers are relatively small and so should not noticeably affect the accuracy of the Destination Distribution Heuristic.

8.3 Possible Improvements

Since the enhanced SWORD detects worm connections passing through a gateway, it can not detect worm connections which are between internal hosts. Therefore, one area of improvement could be related to the detection of internal worm connections. One solution would be to essentially leave the enhanced SWORD as it is and add on a second system such as an ARP-based detection method [30]. Another solution would be to have the system observe all of the internal traffic, in addition to the traffic that crosses the gateway. By observing the internal traffic of the network, we may be able to detect a worm-infected host faster than if we only observe traffic going through the gateway. Researching the advantages and trade-offs of each solution is an area of our future work.

Performance also may be improved by using sliding windows with a finer grain – for instance by having separate windows for each port number. This may decrease the threshold required to differentiate worm connections from normal connections, in turn decreasing the time required for detection.

9. Conclusions

Detecting whether or not an individual host is infected by zero-day worms is critical and must be accurate and fast. Our research shows how the SWORD system can be enhanced to successfully meet this goal.

SWORD is designed to detect zero-day worms at an administrative domain. By enhancing SWORD to also operate at the host level, our research demonstrates that a host infection detection solution does not have to be rooted at individual hosts themselves, but can be cleanly and transparently deployed at the gateway point of the network.

Our contribution in this work is predominantly in the performance evaluation of the enhanced SWORD. By creating synergistic traffic traces through merging real traces with simulated GLOWS worm traces, we are able to evaluate the efficacy and efficiency of the enhanced SWORD in detecting whether individual hosts are infected by zero-day worms. The enhanced SWORD can not only detect hosts infected by different types of worms, but can also detect both high-speed and low-speed worms, all successfully with high accuracy and low latency. Given that the payload is not relevant in the detection process, the results also apply to polymorphic worms that dynamically change or encrypt their payload.

10. Acknowledgments