SWorD: A *S*imple *Wor*m *D*etection Scheme

Matthew Dunlop[1], Carrie Gates[2], Cynthia Wong[3], and Chenxi Wang[3]

[1] United States Military Academy, West Point, NY 10996, USA,
`matthew.dunlop@usma.edu`,
[2] CA Labs, CA, Islandia, NY 11749, USA,
`carrie.gates@ca.com`,
[3] Carnegie Mellon University, Pittsburgh, PA 15213, USA,
{`cynthiaw, chenxi`}`@cmu.edu`

Abstract. Detection of fast-spreading Internet worms is a problem for which no adequate defenses exist. In this paper we present a *S*imple *Wor*m *D*etection scheme (*SWorD*). *SWorD* is designed as a statistical detection method for detecting and automatically filtering fast-spreading TCP-based worms. *SWorD* is a simple two-tier counting algorithm designed to be deployed on the network edge. The first-tier is a lightweight traffic filter while the second-tier is more selective and rarely invoked. We present results using network traces from both a small and large network to demonstrate *SWorD*'s performance. Our results show that *SWorD* accurately detects over 75% of all infected hosts within six seconds, making it an attractive solution for the worm detection problem.

1 Introduction

The problem of worm detection and containment has plagued system administrators and security researchers. Many detection and containment schemes have been proposed. However, few of them have made it into real production systems. This is primarily for two reasons – the possibility of *false positives* and *administration complexity.*

False positives consume valuable resources and help to hide real attacks. In some cases, they result in serious consequences (e.g., loss of business for ISPs that perform automatic filtering). Administration complexity implies cost. Many existing schemes are overly complex to manage, which makes a difficult business case.

In this paper we introduce a **S**imple **Wor**m-**D**etection method called *SWorD*. *SWorD* is meant to be a quick and dirty scheme to catch fast-spreading TCP worms with little complexity. We stress that *SWorD* is not designed to be an all-capable, comprehensive scheme. Rather, the value of *SWorD* lies in its simplicity and good-enough precision, which targets it for immediate deployment.

This work was conducted while the authors were affiliated with CMU.
The views expressed in this paper are those of the author and do not reflect the official policy or position of the United States Government, the Department of Defense, or any of its agencies.

The core of *SWorD* consists of a simple statistical detection module that detects changes in statistical properties of network traffic. More specifically, *SWorD* processes network packets and computes the approximate entropy values of destination IPs for recent traffic. The underlying rationale is that benign traffic typically exhibits a stable range of destination entropies while the presence of scanning worms significantly perturbs the entropy [17]. In *SWorD*, we use a simple counting algorithm to approximate the entropy calculation.

We present an empirical analysis of *SWorD*, based on off-line network traces containing both Blaster and benign traffic. The traces were collected from the border of a network with 1200 hosts and from an Internet service provider containing over 16 million hosts. The analysis results show that *SWorD* is able to detect scanning worms effectively, while maintaining a low false positive rate.

Since *SWorD* uses network statistics to determine infected hosts, it is suitable for deployment at border routers of networks and places where aggregate traffic can be observed. This makes *SWorD* more attractive than other schemes that must maintain state per network host.

The rest of the paper is structured as follows: Sect. 2 covers related work. Sect. 3 outlines the *SWorD* algorithm. Sect. 4 describes the conditions under which *SWorD* was tested and provides results on a small network, while Sect. 5 provides results on a large network. In Sect. 6, we compare *SWorD* with a related algorithm, and conclude in Sect. 7.

2 Related Work

2.1 Automatic Containment

There has been much research in the area of automatic containment of Internet worms. Rate limiting schemes fall into this category. In the area of rate limiting worm defenses, Williamson [21] proposed the idea of host-based rate limiting by restricting the number of new outgoing connections. He further applied this mechanism to email worms by rate limiting emails to distinct recipients [22]. Wong et al. [23] studied the effects of various rate limiting deployment strategies. Chen et al. [3] devised a rate limiting mechanism based on the premise that a worm-infected host will have more failed connections. Our work is different in that rate limiting implemented at the border does not provide detection, while detection at the border is the focus of *SWorD*. By contrast, rate limiting implemented at the host, such as in Williamson's work [21], does provide detection but requires installation at all host sites, rather than a single installation at the border as can be done with *SWorD*.

2.2 Signature Generation

Signature generation schemes hold much promise, but still have difficulty against zero-day worms. Another issue is signature distribution. Earlybird [13], Autograph [9], Polygraph [10], PAYL [19], TREECOUNT and SENDERCOUNT [6],and a vulnerability-based signature scheme by Brumley et al. [2] are examples of signature generation techniques.

2.3 Detection

As mentioned earlier, *SWorD* is best described as a detection algorithm. Threshold Random Walk (TRW) [8], Reverse Sequential Hypothesis Testing ($\overleftarrow{HT}$) [12], Approximate TRW [20], and SB/FB [14] are other examples of detection schemes. In TRW, which focuses on scan detection rather than specifically worm detection, a host is labeled as infected or benign if it crosses a certain upper or lower threshold respectively. A successful connection results in movement toward the lower threshold while an unsuccessful connection results in movement toward the upper threshold. $\overleftarrow{HT}$ and Approximate TRW are variations of TRW. $\overleftarrow{HT}$ uses reverse hypothesis testing combined with credit-based rate limiting to achieve better results than TRW. SB/FB is an adaptive detection scheme that changes based on network traffic. Venkataraman et al. [18] present a detection scheme that uses a streaming algorithm to detect k-superspreaders. A k-superspreader is any host that contacts at least k distinct destinations within a given period. The superspreader technique is most closely related to our work and examined in more detail in Sect. 6.

3 Detection Algorithm

SWorD is a simple statistical detection tool used to identify fast-spreading worms. Since these worms do so much damage so quickly, it is important to have a mechanism on the network edge that can detect and filter them. *SWorD* detects fast-spreading worms by computing a quick count of connection attempts, flagging those hosts that attempt more connections than what is deemed "normal." *SWorD* can be used on outbound traffic to identify and filter internal hosts that are misbehaving (worm-infected or rapidly scanning), as well as on inbound traffic to identify and filter external hosts that might be infected.

In this section, we present *SWorD*'s two-tiered detection algorithm. The first-tier is a "sliding window counting" algorithm that identifies traffic anomalies. If the first-tier count reaches a certain threshold, the second-tier algorithm is invoked, which is used to pinpoint and automatically filter the hosts responsible for the anomalous behavior. Since *SWorD* uses automatic filtering, a host will not trigger multiple alarms due to subsequent traffic. Figure 1 outlines the parameters used for *SWorD*.

w	sliding window size
D	first-tier threshold of distinct destination IPs
S	second-tier threshold of distinct source-destination IPs pairs

Fig. 1: Parameters.

3.1 Algorithm

In the first-tier algorithm, we keep a sliding window holding the destination IP addresses of the last w outgoing connection attempts (TCP SYN packets) from the monitored network. For each sliding window, we count the number of distinct

```
first-tier(){
    for(each outgoing SYN packet)
        /* remove oldest packet from window and adjust count for dest IP */
        dst_IP(oldest_SYN)- -
        POP oldest_SYN from SLIDING_WIN

        /* add next SYN packet to window and adjust count for dest IP */
        PUSH new_SYN onto SLIDING_WIN
        dst_IP(new_SYN)++

        if(UNIQUE_DST_COUNT/window size > D)
            second-tier(UNIQUE_DST_COUNT)}
```

Fig. 2: first-tier sliding window counting algorithm

```
second-tier(UNIQUE_DST_COUNT){
/*COUNT distinct dst IPs in SLIDING_WIN for last source added to window*/
    for($i \leftarrow 0$ to $w$)                    /* check each packet in window */
        /* if src-dst pair unique */
        if(SLIDING_WIN[$i$].src_IP = src_IP(new_SYN) AND
          SLIDING_WIN[$i$].dst_IP not in UNIQUE_DST_IPs)
            add SLIDING_WIN[$i$].dst_IP to UNIQUE_DST_IPs
            increment SRC_DST_COUNT
    if(SRC_DST_COUNT/UNIQUE_DST_COUNT > S)
        FLAG src_IP}
```

Fig. 3: second-tier *find_scanner* algorithm

destination IPs. If this number is over a certain threshold (D), the second-tier algorithm will be invoked. The first-tier algorithm is described in Fig. 2.

The second-tier algorithm (see Fig. 3) identifies the specific host exhibiting scanning behavior typical of a fast-spreading worm. This tier should rarely be invoked during normal operation. We assume that the goal of a fast spreading worm is to infect large portions of the IP space rapidly, and so will scan a large number of distinct IPs in a small time period. In the context of our algorithm, this translates to a specific source address occupying a larger than average portion of the sliding window. Therefore, we count the number of distinct destinations contacted by the newest SYN packet. If this number divided by the total number of distinct destinations in the window exceeds our threshold, we flag and filter the host. Note that, as our algorithm only operates on SYN packets required to establish a TCP connection, we do not interfere with pre-existing connections.

3.2 Extensions

The basic algorithm as stated above is effective at identifying worm-infected and scanning hosts, but it also introduced false positives. In order to prevent these false positives, we introduce two extensions to the basic algorithm: *burst credit* and *whitelist*.

Burst Credit The basic algorithm can not easily distinguish between a bursty client and a scanning worm within a short period of time. One way bursty traffic differs from a scanning worm is that a scanner typically does not contact the same machine repeatedly. On the other hand, a normal user client will likely contact the same destination address multiple times [11], leading to a number of nondistinct source-destination address pairs. To make allowances for bursty clients, we use a technique we call *burst credit*. For each destination port 80 SYN packet, we subtract one from the distinct destination address count for every nondistinct destination contacted by the same source. Since only the most recent SYN packet's port information is checked, there is no additional state maintained. Note that this extension can be applied to other ports that experience bursty traffic. In this work, we consider bursty web clients only.

An attacker can attempt to "game" this extension by devoting 50% of her packets to nondistinct destination addresses. However, this is only possible on bursty ports. It is not possible for an attacker to disguise a worm attacking another port by flooding the network with nondistinct connections to a bursty port as no port information is stored.

Whitelist There are some hosts, such as mail servers, that exhibit behavior that could cause them to be falsely flagged. To prevent this from happening, we added a *whitelist* extension. By this extension, any host in the whitelist would be ignored by the second-tier algorithm.

3.3 Storage and Computational Cost

Storage Cost Our first-tier algorithm must maintain both source and destination IP addresses for second-tier processing. Since each IP address pair (source and destination IP) is 8 bytes, the space requirement for the sliding window is $8w$ bytes where w is the window size. We utilize a hash map with a simple uniform hash function and a load factor of 0.6 [4] to track and count the distinct destination addresses. This adds $10w$ bytes to the storage requirement. In the second-tier algorithm, we use a hash set (since we need only check presence of the address) with a simple uniform hash function and a load factor of 0.6. At the worst case this adds another $6w$ bytes to the storage requirement, bringing the total storage requirement for the first and second tier algorithms to $24w$ bytes.

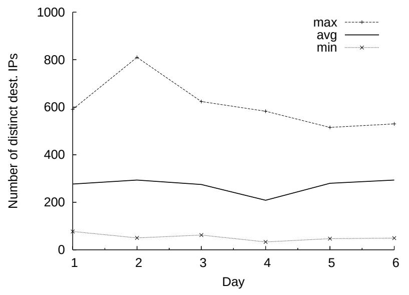


Fig. 4: Maximum, minimum, and average number of distinct IPs seen daily in w consecutive SYN packets for $w = 1000$ using the small network trace.

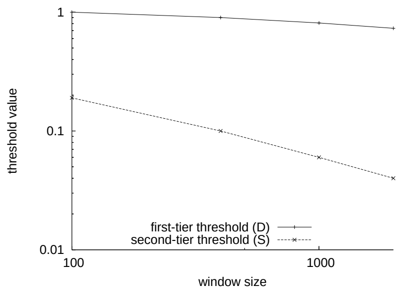


Fig. 5: Power law plot of empirically chosen thresholds versus window size using the small network trace. Window sizes from 100 to 2000 are shown on the x-axis and values for D and S are shown on the y-axis.

Computational Cost The computational cost for the first tier includes two hash lookups per SYN packet at $O(1)$ for each hash lookup. For n packets seen in a specified time period, this results in a linear computational expense of roughly O(n). The second-tier involves one hash lookup for every connection in the window for counting distinct destinations, O(w). If we use p as the probability of entering the second-tier, the cost for the second-tier algorithm is O(pnw) for processing each packet in the specified time period. This brings the total expense for *SWorD* to O(n+pwn). As the window size is a fixed value, the computational expense remains linear. We show in Sect. 4.2 that p is small during periods of uninfected traffic, so that actual expense is close to O(n).

3.4 Parameter Selection

The values that we use for the first- and second-tier thresholds are empirically derived. We use two different networks (described in Sect. 4 and 5) in our evaluations. Traffic collected from the smaller network (Sect. 4) is used to derive equations for determining appropriate thresholds. These equations are then used to determine thresholds for a large network (Sect. 5), demonstrating their effectiveness given a very different network traffic level, size and topology.

First-tier Threshold Selection Since the first-tier threshold is the number of distinct destinations allowed in a window before triggering the second-tier algorithm, choosing the right threshold is particularly important. If the threshold is set too low, *SWorD* will enter the heavier weight second-tier function unnecessarily during normal operations. It is also likely to result in a higher false positive rate. If the threshold is set too high, an increased detection time and false negative rate may ensue.

In order to decide how to set the threshold, we monitored the number of distinct IPs seen in each w consecutive SYN packets during normal operations using the small network trace. Figure 4 shows the daily maximum, as well as the average and the minimum, number of distinct destinations seen for w consecutive outbound SYN packets, where $w = 1000$. The first-tier threshold, D, was chosen to be within $\frac{\sigma}{4}$ of the total maximum value seen during the six-day period, where σ is the standard deviation (We found values larger than $\frac{\sigma}{4}$ away from the maximum produced increasingly less accurate results the farther away from the maximum we moved.). This same technique for selecting D was applied to $w = 100$, 400, and 2000.

Figure 5 shows the first-tier threshold value versus window size on a log-log plot. The relationship appears to be power law. Using linear regression, we developed the following equation for D.

$$D = e^{(0.63-0.12\ln w)} \tag{1}$$

In the remainder of this paper, we will use Eq. 1 to estimate D for different window sizes, w, for both test networks.

Second-tier Threshold Selection Recall a source IP is flagged as infected if the number of distinct destination IPs contacted by that source exceeds the second-tier threshold, S. We use a similar technique to that described in the previous section to determine S. Whenever the algorithm enters the second-tier, we examine what percentage of all the distinct destinations in the sliding window were contacted by each benign source. (Note that to have the algorithm enter the second-tier, we need to use network traffic collected during infection.) We set S just above this percentage to avoid mislabeling benign sources. The values we empirically selected for S also follow a power law relationship with w (see Fig. 5). As before, we used linear regression to produce an equation for computing S.

$$S = e^{(1.11-0.57\ln w)} \tag{2}$$

We will show in Sect. 5 that Eq. 1 and 2 are generalizable to other networks.

Sliding Window Selection Since we can tune our first and second tier thresholds based on sliding window size, window size selection is not as critical. We do, however, want to choose the smallest practical sliding window size to reduce the storage and computational expense. However, the window needs to be large enough to provide adequate sampling of the network traffic. Thus the window size is related to the volume of traffic observed at the border router.

4 Results on a Small Network

The experiments presented in this section and in Sect. 6, along with the parameter selection described in Sect. 3.4, were conducted using traffic traces collected

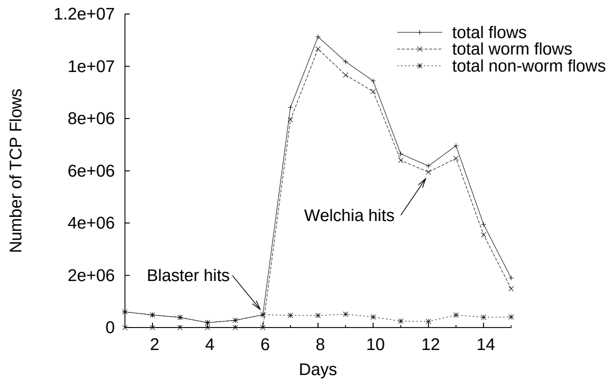


Fig. 6: Number of outbound TCP flows at the edge router per day for the small network Blaster/Welchia trace.

from the edge router of a 1200 host academic network. The network serves approximately 1500 users. Since May 2003 we recorded TCP packet headers leaving and entering the network. During the course of tracing, we recorded two worm attacks: *Blaster* [15, 1] and *Welchia* [16]. For each attack recorded, we conducted post-mortem analysis to identify the set of infected hosts within the network.

For the purpose of this analysis, we use a 15-day outbound trace, from August 6th to August 20th, 2003. This period contains the first documented infection of Blaster in our network, which occurred on August 12th.

Figure 6 shows the daily volume of outgoing traffic as seen by the edge router for the trace period. As shown, the aggregate outgoing traffic experienced a large spike as Blaster hit the network on day 7. At its peak, the edge router saw over 11 million outbound flows in a day. This is in contrast to the normal average of 400,000 flows/day. The increase in traffic is predominantly due to worm activity.

Our implementation of *SWorD* included the two extensions described in Sect. 3.2. For the experiments on this network trace, we gave *burst credit* to destination port 80 and we whitelisted one internal mail server. Additionally, we analyzed the outbound traffic because we have exact information on internally infected hosts.

4.1 Accuracy

To measure the accuracy of *SWorD*, we use false positive (FP) and false negative (FN) rates. The false positive rate is the percentage of benign hosts misidentified as infected. The false negative rate is the percentage of infected hosts not identified by *SWorD*. For the small network, the total daily number of benign and infected hosts is shown in Table 1.

Table 2 gives the average FP and FN rates for *SWorD* using different sliding window sizes. Results are broken down in terms of pre-infection and post-

Day	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Benign	759	769	760	690	638	736	709	690	686	574	661	656	738	731	812
Infected	0	0	0	0	0	0	57	38	34	30	15	11	17	16	7

Table 1: Number of benign hosts during each day (Benign) and the number of known infected hosts during each day (Infected) using the small network trace data.

		Pre-inf.(%)	Post-inf.(%)
$w = 100$	FP Rate	0.044	0.174
	FN Rate	0	0
$w = 400$	FP Rate	0.022	0.113
	FN Rate	0	0.195
$w = 1000$	FP Rate	0	0.123
	FN Rate	0	0.889
$w = 2000$	FP Rate	0	0.139
	FN Rate	0	0.195

Table 2: Comparison of average pre and post infection FP/FN rates for different window sizes (for window sizes 100, 400, 1000, and 2000, $D = 0.99$, 0.90, 0.82, & 0.74 and $S = 0.19$, 0.10, 0.06, & 0.04 respectively).

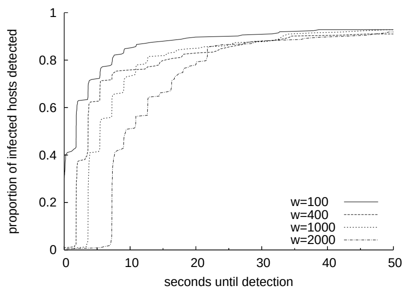


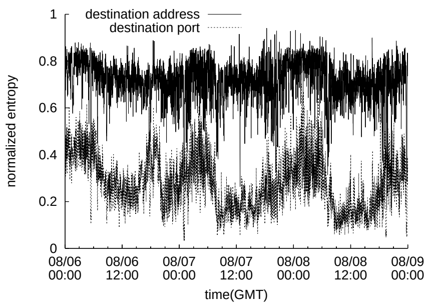
Fig. 7: Cumulative distribution of the number of seconds needed to detect an infected host for *SWorD* using four different sliding window sizes. Time is counted from the first infected packet that enters the network from each infected host. Results are collected over the period of infection (days 7-15).

infection. The average FP rate for all window sizes never exceeded 0.05% during the pre-infection period and 0.2% during the post-infection period. For $w = 100$, the FN rate was zero. Larger window sizes did have false negatives, but the FN rate did not exceed 0.9% over the eight day post-infection period. It is possible to select parameters such that we detect all infected hosts. However, the tradeoff is a higher number of false positives.

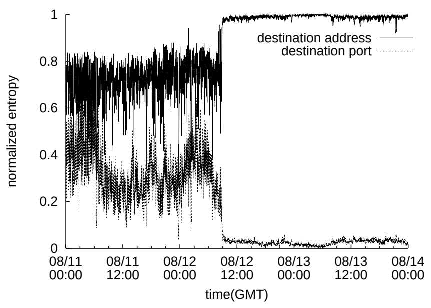
For this data set, we had at most three false positives in any given day from an average of 722 active hosts. Throughout the entire 15-day trace, there were a total of seven hosts misidentified as infected. Examining the behavior of these hosts showed that they were detected primarily due to peer-to-peer traffic.

4.2 Timeliness of Detection

Figure 7 shows the proportion of infected hosts detected over time by window size. Notice that for a sliding window size of 2000, over 78% of infected hosts are detected within 20 seconds. A sliding window size of 100 detects approximately the same number of infected hosts within six seconds. Using smaller sliding window sizes results in quicker detection time as well as reduced storage cost and second-tier computational expense.



(a) Traffic prior to Blaster infecting the small network.



(b) Results as Blaster is infecting the small network.

Fig. 8: Plots of normalized destination address and destination port entropy from our small network trace data.

Time spent in the second-tier algorithm also contributes to the timeliness of detection. Over our six day "pre-outbreak" trace period, almost 2.5 million SYN packets were observed. However, the second-tier algorithm was invoked only 1298 times for $w = 100$, 63 times for $w = 400$, and not at all for $w = 1000$ and $w = 4000$. These results suggest that the probability of entering the second-tier, p, is approximately 0.05% during normal operations. As expected, the second-tier was invoked more often during the outbreak period.

5 Results on a Large Network

Our second data set is from a large Internet Service Provider (ISP) servicing more than 16 million hosts. During the Blaster attack, the ISP's network received a large volume of inbound infection attempts. The network was not infected by Blaster internally due to very restrictive port filtering (which included port 135). We analyzed inbound traffic as the network received a large volume of inbound infection attempts, while no internal hosts were infected. For this network trace, we gave *burst credit* to destination port 80, however we did not use a whitelist.

Unlike the small network data set described in Sect. 4, we do not have a list of known infected (external) hosts for the incoming ISP network trace. To determine when the network began seeing Blaster infected packets, we use a network entropy detection scheme very much like the one by Valdes [17].

Entropy-based Detection Valdes [17] observed that normal network traffic attributes (e.g., destination IPs, ports, etc.) follow a predictable entropy pattern unique to the behavior of that network. Anomalous traffic on the same network will cause a change in the entropy pattern and can be a sign of infection.

As a proof-of-concept, we implemented a variation of Valdes's algorithm and applied it to the traces obtained from the small network. To establish a baseline for normal network traffic, we analyzed outbound network flows for a period of three days prior to the outbreak of Blaster. The graph in Figure 8(a) shows that despite fluctuation (e.g., diurnal patterns, weekend versus weekday patterns), the destination address and port entropy levels fall within a relatively stable and predictable range.

Figure 8(b) shows the same entropies when Blaster hit the network. We see that the destination IP entropy, after Blaster infects the network, is very close to one. An entropy value of one indicates a completely random sample. This is consistent with Blaster behavior as it attempts to contact unique destinations to achieve a large fan-out. The destination port traffic exhibits a decrease in entropy as Blaster hit. Again, this is in line with Blaster behavior. As a larger portion of the traffic mix becomes horizontal-scan traffic on the same port, port-entropy decreases. It is also worth noting that when the network becomes infected with worm traffic, the variance of the entropy decreases. This characteristic is present in both our results and those of Valdes [17]. This is expected since worm flows follow similar traffic patterns and the volume of worm flows overwhelm well-behaving flows.

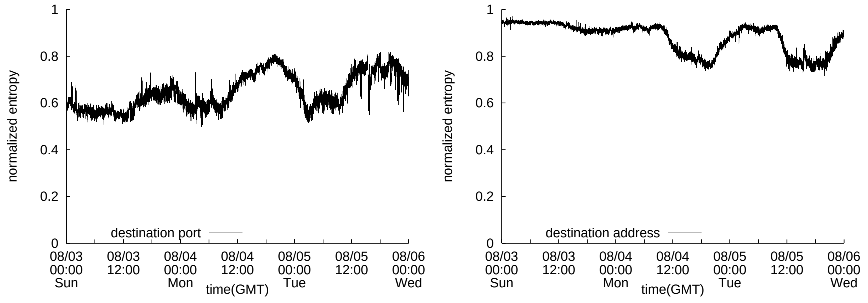
5.1 Experiment Set-up

In order to determine the presence of infected traffic inbound to the ISP, we combined the normal traffic with filtered traffic based on an access control list (ACL) and calculated the traffic entropy. Figure 9 shows the resulting destination address and port entropy graphs for both uninfected and infected traffic including ACL traffic. Figure 9(b) illustrates that destination port entropy dropped and destination address entropy increased just before 18:00 on the 11th of August. At this time, the network saw a sharp increase in the volume of destination port 135 traffic, which is indicative of Blaster. Note that the destination address entropy illustrates the near to completely random nature of destinations contacted. As a result, when Blaster hits, as shown in Fig. 9(b), we do not see a drastic change, but rather a small jump back to maximum entropy rather than the gradual daily fluctuation likely due to work cycle.

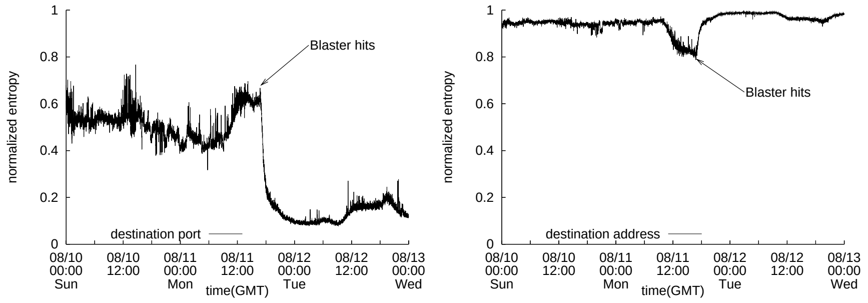
To run *SWorD* on the ISP data, we chose a sliding window of size 200,000. (This is comparable to using a window size of 50 for the small network in Sect. 4.) We used Eq. 1 and 2 to select values for the first and second-tier thresholds, resulting in $D = 0.43$ and $S = 0.003$.

5.2 Results Using *SWorD*

After using *SWorD* to filter out suspected infected hosts, we ran the entropy-based algorithm on the remaining network traffic. From Fig. 10, we see that the normalized destination address and port entropy post-*SWorD* no longer displays the network anomalies seen in Fig. 9(b). Notice that as well as filtering out the network anomaly caused by the increased volume of destination port 135 traffic, the entropy values before the infection are consistently higher for destination port and lower for destination address than those in Fig. 9. We attribute this



(a) Normalized entropy of traffic prior to Blaster hitting the ISP.



(b) Normalized entropy results as Blaster hits the ISP.

Fig. 9: Plots of normalized destination port and destination address entropy from inbound ISP flow traffic including all ACL filtered traffic.

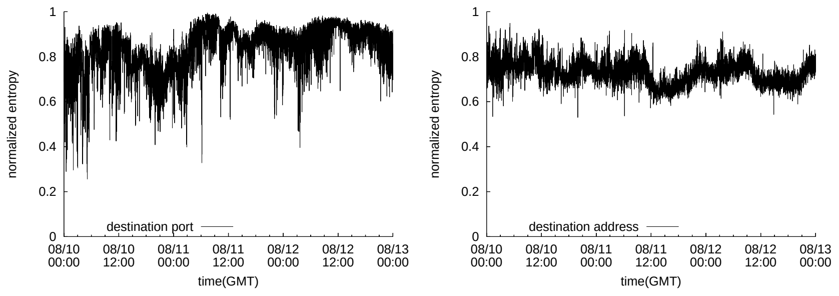


Fig. 10: Plot of normalized destination port and destination address entropy with *SWorD* performing automatic filtering on the inbound ISP flow traffic (including ACL filtered traffic) during the period of infection.

phenomenon to the presence of other scan traffic contained in the ACL filtered traffic. Figure 11 illustrates that the normalized entropy of the network traffic

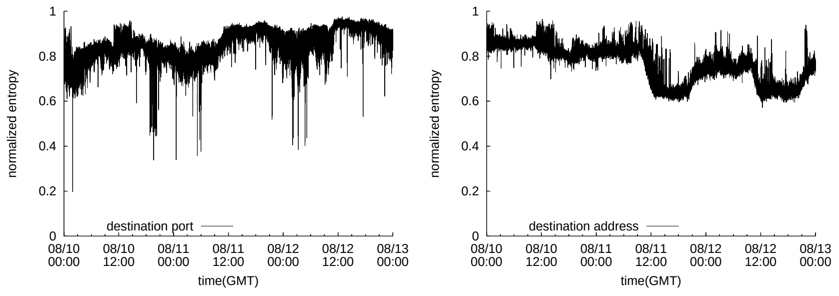


Fig. 11: Plot of normalized destination address and destination port entropy of inbound ISP flow data *excluding* all ACL filtered traffic.

excluding ACL filtered traffic follows the same pattern as what we achieved using *SWorD*.

5.3 Accuracy

The entropy-based results give us an indication that *SWorD* is filtering out some malicious network traffic, but do not allow us to conclude how accurate our results are. Namely, we cannot be sure if *SWorD* is filtering out all of the infection, or if it is filtering out legitimate traffic. As a second tool to help determine the accuracy, we randomly select three different post-infection hours and examine some of the network traffic characteristics. To provide us with an idea of how *SWorD* is doing in terms of false negatives, we analyze the unfiltered traffic to see if it contains any of the hosts in the ACL list[1]. We found that *SWorD* successfully filtered all hosts contained in the ACL list during the three hours.

Determining how *SWorD* is performing in terms of false positives is more difficult. The ISP commonly sees a large number of inbound hosts conducting scanning. Since fast-spreading worms perform scans to propagate, when we flag a host as infected we cannot determine if the cause is specifically due to worm behavior. Therefore, a coarse method is required to determine how many of the hosts *SWorD* filtered out were worm-infected or scanning hosts. The coarse method we used was to analyze the number of SYN-only connections made by the set of *SWorD*-filtered hosts as compared to the overall number of connections made by the set of *SWorD*-filtered hosts. A "SYN-only connection" is defined as a single packet flow with no flags set other than the SYN flag. We found that out of over 130 million *SWorD*-filtered connections in an hour, over 95% were SYN-only connections. Examining the hosts in the *SWorD*-filtered set showed that out of over one-million *SWorD*-filtered hosts, over 97% were making SYN-only connections. From this course measure, we estimate roughly a 3% false positive

[1] Any potential Blaster connections would be included in the ACL list, since it contains all flows attempting to connect to destination port 135.

rate. There is likely some fluctuation in this estimate. For example, it is possible that we filtered a legitimate host and that the host made SYN-only connections. On the other hand, it is possible that a malicious host contacting a hit list did not make any SYN-only connections, as it may have tried infecting hosts that respond to its SYN requests.

5.4 Timeliness of Detection

Since we do not have exact information on infected hosts, we can not use the same method for determining the time to detect an infected host as we do with the small network. Instead, we refer to our timeliness results for the small network with $w = 100$ (see Fig. 7) to predict a null hypothesis for the large network. Our null hypothesis is that 60% of infected hosts will be detected within three seconds. We then randomly selected 25 hosts that contacted destination port 135 during the Blaster infection period. We compared the time each of these hosts sent out the first port 135 connection attempt to the time *SWorD* flagged the host. We found that all 25 hosts were flagged within one second. Given the null hypothesis that 60% will be detected in under three seconds, the probability that we would observe 25 of 25 detected in under three seconds is 0.00028%. We therefore reject our null hypothesis in favor of an alternative hypothesis that greater than 60% will be detected in under three seconds.

6 Comparison with a Related Scheme

In this section, we compare *SWorD* to Superspreader [18]. We chose Superspreader because, similar to *SWorD* its goal is to detect fast-spreading hosts. In addition, Superspreader and *SWorD* are both deployed on the network edge and neither maintain per-host statistics. One major way the two schemes differ is that Superspreader uses sampling whereas *SWorD* does not. For our comparison, we implemented the Superspreader one-level filtering algorithm using a sliding window. More details on the Superspreader algorithm can be found in the Superspreader paper [18].

6.1 Parameter Selection

For a host to be identified as a k-superspreader, it must contact at least $\frac{k}{b}$ distinct destinations within a window of size W. By definition, k is the number of distinct destinations a host can contact before being considered a superspreader, while b is a constant designed to scale k according to the amount of sampling being done. In order to identify Superspreaders in a timely manner, a host is labeled as a Superspreader after it contacts $\frac{k}{b}$ distinct destinations.

The Superspreader paper does not discuss parameter selection in specific detail. Therefore, we devise a method to choose these values based off the volume of infected traffic we observe from the small network. To select values for k and b, we computed the number of packets each infected host had in a window of size W during the infection period (days 7-15). We set k equal to the average of these counts over the entire infection period, which we refer to as "total avg." For calculating b, we calculated the daily average number of packets from the infected hosts. We then took the minimum of these daily averages, referred to as "min daily avg," and set b=(total avg)/(min daily avg). As in the superspreader paper, we used an error rate of $\delta = 0.05$. We experimented with other values of k and b that were one and two standard deviations away and found the best results using the values we calculated [5].

For our first comparison of Superspreader to *SWorD*, we used $W = 2000$, $k = 337$, and $b = 2$. With these parameters, the sampling rate, $\frac{c_1}{k}$, is equal to 0.25. Note this sampling rate is higher than those used in their paper, which should only benefit the results achieved by Superspreader in terms of detection time.

6.2 Accuracy

Accuracy based on definition of a *k*-superspreader [18]. By definition of a superspreader, a host is identified as a k-superspreader regardless of whether or not the host is actually infected. In light of this, we first ran the superspreader algorithm on our small network trace data based solely on the definition of a superspreader. According to the superspreader paper, a FP is any host that contacts less than $\frac{k}{b}$ distinct destinations but is labeled as a superspreader. A FN is any host that contacts more than k distinct destinations, but does not get labeled as a superspreader. According to this definition, our results showed zero FNs and one FP throughout the 15-day run using $W = 2000$.

In general, it is not possible to select parameters for *SWorD* that flag these same sets of hosts. The main reason for this is that *SWorD* is not designed to flag a host unless the "normal" entropy of network traffic is perturbed, thus indicating an outbreak. The consequence of this is that *SWorD* may not flag an individual scanning host because it does not dominate enough of the network traffic to overcome the effect normal traffic has on the network. This is desirable in the sense that *SWorD* focuses on identifying infected hosts from a worm outbreak, rather than identifying scanners (which can be detected using other algorithms, such as TRW[7]). *SWorD* also thus avoids entering the second-tier algorithm unless a dominating fast-scanning host is present, thus increasing its speed. Superspreader, on the other hand, is designed to identify every host that connects to more than $\frac{k}{b}$ distinct destinations.

Accuracy based on real infected trace data. Using the parameters we selected in Sect. 6.1, we compared how effective the Superspreader algorithm was at detecting the infected worm traffic from our small network trace data. We compared these results against *SWorD* using the same window size. Superspreader was able to detect all infected hosts as opposed to one host missed by *SWorD*. However, on average the FP rate for Superspreader was over 25 times that of *SWorD*. A daily comparison of FP rates is shown in Fig. 12. An analysis

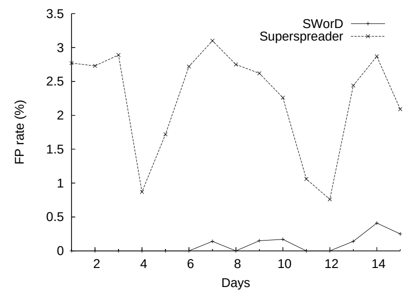


Fig. 12: False positive rates for $SWorD$ and Superspreader based off the small network trace data infected by Blaster. Both algorithms are using a sliding window size of 2000.

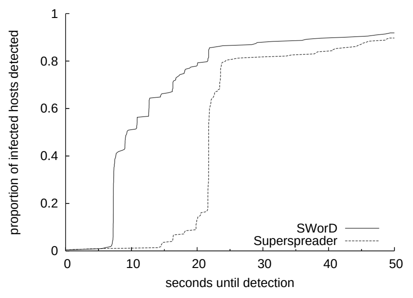


Fig. 13: Cumulative distribution of the number of seconds needed to detect an infected host for $SWorD$ and Superspreader using a sliding window size of 2000. Results are collected over the period of infection (days 7-15).

of the hosts Superspreader mislabeled as infected showed that over 85% were the result of bursty web traffic, and so providing Superspreader with a mechanism for detecting bursty web traffic (as added to $SWorD$) will remove this source of false positives. Of those remaining, all but one mislabeled host was the result of peer-to-peer traffic. We found no evidence that the remaining mislabeled host was malicious.

6.3 Storage Requirement

The Superspreader paper [18] does not discuss the storage required for their algorithm under the sliding window scheme. However, we can estimate the storage requirement. The algorithm must maintain the source and destination IPs for all packets in the window. The storage requirement for this is $8W$ bytes. For the non-sliding window version of the algorithm, there is a 3-byte requirement per source IP. Since there are at most W sources in the sliding window version, we add another $3W$ to the storage cost. This brings the total storage requirement to $11W$ bytes.

Comparing the $11W$-byte storage requirement for Superspreader to the $24w$-byte storage requirement for $SWorD$(see Sect. 3.3), we see that $SWorD$ requires over twice the storage of Superspreader when using similar window sizes. However, $SWorD$ does not require windows that are as large as those of Superspreader [5].

6.4 Timeliness of Detection

Comparing $SWorD$ and Superspreader in terms of time until each infected host is detected demonstrates that on average $SWorD$ detects infected hosts faster. Figure 13 shows the proportion of infected hosts detected over time for both algorithms. For $SWorD$, 88% of all infected hosts were detected within 30 seconds as opposed to 83% for Superspreader. Of the 88% that *SWorD* detected, 51% were detected within the first 10 seconds. By 20 seconds, *SWorD* detected 78%. Superspreader only detected 13% by 20 seconds. Recall from Fig. 7 that *SWorD* performs even better with smaller window sizes.

Compared to Superspreader, *SWorD* is able to achieve faster detection with higher accuracy. This is primarily because *SWorD* uses smaller window sizes and does not require sampling. Since *SWorD* does not use sampling, it has a higher storage requirement than Superspreader when comparable window sizes are used. However, we have shown that when parameters are chosen to maximize accuracy, *SWorD* requires less storage. A more detailed comparison of the two algorithms is available [5].

7 Conclusion

In this paper we presented a technique for detecting and automatically filtering fast-spreading worms and scanners. Our algorithm is simple to implement and effective. By bounding the storage and computation overhead, we make deployment on the network edge feasible.

We tested *SWorD* on both a small and a large network. On the small network, we showed that our algorithm is able to quickly detect worm infected hosts – 78% within six seconds. We also demonstrated that *SWorD* is able to achieve these results with high accuracy – zero FNs and an average FP rate of 0.1%, where our FP rate is based on the number of hosts observed on any given day, rather than on the traffic volume. Our results from applying *SWorD* to a large ISP with over 16 million hosts indicate that its effectiveness is not limited by network size or traffic direction. For example, *SWorD* successfully detects *all* Blaster infected hosts. Taking a random sampling of these hosts, we find detection occurs within one second of the first infected packet.

References

1. M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The blaster worm: Then and now. *IEEE Security and Privacy Magazine*, 3(4):26–31, July-Aug. 2005.
2. D. Brumley, J. Newsome, D. Song, and S. Jha. Towards Automatic Generation of Vulnerability-Based Signatures. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2006.
3. S. Chen and Y. Tang. Slowing down Internet worms. In *Proceedings of 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.
4. Cormen, Leiserson, and Rivest. *Introduction to Algorithms.* MIT Press, Cambridge Mass., first edition, 1990.
5. M. Dunlop. Anomaly detection in network traffic and automatic filtering. Master's thesis, Carnegie Mellon University, 2006.

6. P. Gopalan, K. Jamieson, P. Mavrommatis, and M. Poletto. Signature metrics for accurate and automated worm detection. In *WORM '06: Proceedings of the 4th ACM workshop on Recurring malcode*, pages 65–72, New York, NY, USA, 2006. ACM Press.
7. J. Jung, V. Paxon, A. W. Berger, and H. Balakrishman. Fast portscan detection using sequential hypothesis testing. In *Proceedings of 2004 IEEE Symposium on Security and Privacy*, 2004.
8. J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.
9. H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, San Diego, California, USA, August 2004.
10. J. Newsome, B. Karp, and D. Song. Polygraph, Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.
11. F. Qiu, Z. Liu, and J. Cho. Analysis of user web traffic with a focus on search activities. In *the 8th International Workshop on the Web & Databases (WebDB)*, pages 103–108, June 2005.
12. S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *Recent Advances in Intrusion Detection (RAID) 2004*, France, September 2004.
13. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation*, December 2004.
14. A. Studer and C. Wang. Fast detection of local scanners using adaptive methods. In *Applied Cryptography and Network Security (ACNS) 2006*, Singapore, June 2006.
15. Symantec. W32.Blaster.Worm. World Wide Web, http:// securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html.
16. Symantec. W32.Welchia.Worm. World Wide Web, http:// securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html.
17. A. Valdes. Entropy characteristics of propagating Internet phenomena. In *The Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, September 2003.
18. S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Proceedings of the* 12^{th} *Network and Distributed System Security Symposium (NDSS)*, February 2005.
19. K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Recent Advances in Intrusion Detection (RAID) 2005*, pages 227–246, Seattle, WA, USA, September 2005.
20. N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, 2004.
21. M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002.
22. M. M. Williamson. Design, implementation and test of an email virus throttle. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.
23. C. Wong, C. Wang, D. Song, S. M. Bielski, and G. R. Ganger. Dynamic quarantine of Internet worms. In *Proceedings of DSN 2004*, Florence, Italy, June 2004.