Enterprise Security: A Community of Interest Based Approach

Patrick McDaniel* mcdaniel@cse.psu.edu
Subhabrata Sen† sen@research.att.com
Oliver Spatscheck† spatsch@research.att.com
Jacobus Van der Merwe† kobus@research.att.com
Bill Aiello‡ aiello@cs.ubc.ca
Charles Kalmanek† crk@research.att.com

Abstract

Enterprise networks today carry a range of mission critical communications. A successful worm attack within an enterprise network can be substantially more devastating to most companies than attacks on the larger Internet. In this paper we explore a brownfield approach to hardening an enterprise network against active malware such as worms. The premise of our approach is that if future communication patterns are constrained to historical "normal" communication patterns, then the ability of malware to exploit vulnerabilities in the enterprise can be severely curtailed. We present techniques for automatically deriving individual host profiles that capture historical communication patterns (i.e., community of interest (COI)) of end hosts within an enterprise network. Using traces from a large enterprise network, we investigate how a range of different security policies based on these profiles impact usability (as valid communications may get restricted) and security (how well the policies contain malware). Our evaluations indicate that a simple security policy comprised of our Extended COI-based profile *and* Relaxed Throttling Discipline *can effectively contain worm behavior within an enterprise without significantly impairing normal network operation.*

1 Introduction

The outbreak of the *witty* worm [6], which ironically spread through vulnerabilities in commercial desktop security software, illustrates the importance of multi-facet security countermeasures. This was arguably the first widespread worm to have a truly malicious payload. It also appears that the release of the worm was coordinated across many hosts, making it nearly impossible to identify its origin. Because the desktop security mechanisms were totally compromised, little could be done to prevent infection. The perimeter defenses (e.g., firewalls) only marginally aided the suppression of these worms because of the difficulty of defining and implementing perimeter defenses.

This lack of security is particularly troublesome in the context of corporate enterprise networks. While the Internet itself is becoming an increasingly important part of today's economy, corporate enterprise networks still carry the vast majority of mission critical communications. Therefore, a successful worm attack within an enterprise network will be substantially more devastating to most companies than attacks on the Internet. Furthermore, worms can spread in enterprise networks even if they are completely isolated from the Internet. For example, worms might be introduced by laptops that are used both outside and within the enterprise, or by unauthorized software installations within the enterprise. These attacks are exacerbated by the sheer size of enterprise networks: enterprise networks can span multiple continents, connect tens of thousands of machines, and be used by hundreds of thousands of employees.

The goal of our work is to improve the protection against worms and similar security threats within enterprise networks. While these networks can be quite large they do provide certain properties which make it easier to protect them. In particular they typically have a known network topology, have knowledge of all end hosts allowed on the network, can control the configuration on all routers and switches within the network and can install software on every host deployed in the enterprise. We denote all hosts within the enterprise network in question as *internal* hosts and all other hosts as *external* hosts. The standard role of firewalls is to protect internal hosts from potentially malicious external hosts that typically translate into rules for dropping or passing packets originating from external hosts destined for internal hosts. We acknowledge that even internal nodes may become malicious when they are compromised. Thus each internal node should also be protected from other internal hosts. In the simplest instantiation of our framework, this translates into a set of rules for dropping or

*SIIS Laboratory, CSE, Pennsylvania State University
†AT&T Labs – Research
‡Computer Science, University of British Columbia

allowing packets where both the origin and destination are internal hosts. In the most general case, the set of rules may define an arbitrary subset of the 4-tuple defined by origin and destination IP addresses, protocol, and destination port number where the origin and destination IP addresses are for internal hosts. The design space for such a set of rules is defined along three principle axes: security, usability, and manageability.

Enterprise policies must be manageable to be usable in real networks. Hence, we restrict ourselves to simple policies throughout. Moreover, for all the policies, we describe methods for automatically populating the rules defining the allowed internal-to-internal communication. The primary focus of the paper is an analysis of the usability and security, and the tradeoffs between them, of the chosen set of manageable policies.

If one envisions designing and deploying an enterprise network, including all hardware and software, from the ground up, it may be possible to design and enforce quite rigid internal-to-internal communication policies. But our work does not require such a greenfield approach. Rather our work is aimed at a brownfield environment–an existing large, complex enterprise network. In such an environment, it is (in our experience) nearly impossible to impose rules after the fact without severely affecting usability. This is due both to the diversity of user requirements and to the complexity of explicit and implicit host communications. Moreover, it may be nearly impossible to "reverse engineer" by hand the implicit, existing enterprise-wide policy–even with many hands.

In this paper, we present methods for automatically generating the policies based on several weeks of training data. The essential premise of this approach is that if future communication patterns are constrained to historical "normal" communication patterns, then the ability of malware (e.g., worms) to exploit vulnerabilities in the enterprise is severely curtailed. Of course, such a history-based approach may hinder both usability and security. For usability, this approach may block a perfectly legitimate communication if it didn't occur in the training period, which is undesirable. To handle such possibilities, our policies have two components. The first is a *profile*. This is a set of rules defining internal-to-internal packets that are allowed. Several of the policies we introduce allow for a specified rate of out-of-profile communication. The rate, and the action to take when the rate is exceeded is given by the *throttling discipline (TD)*, a second component of our policies. Our history-based approach may also compromise security if illegitimate communication is part of the training set. We assume that after creation, the profile is scrubbed for illegitimate rules based on the behavior of known worms. We rely on multiple heuristics introduced in our previous work [16] to reduce the amount of illegitimate rules before the profiles are derived.

We should make a brief note here about enforcement of our policies. Our internal-to-internal communication policies can be enforced at either the origin or the destination, or the switch or hub just upstream or downstream, respectively, and the work presented in this paper is agnostic to that decision. However, the motivating perspective of the paper, i.e., of viewing enterprise hosts as potentially malicious, suggests that enforcement should be as close to the potentially malicious entity as possible, i.e., at the originating hosts or the upstream switch. This is consistent with the accepted security practice of deploying the enforcement as close to the source as possible. As a specific example of the advantage of such a placement, several fast spreading worms such as Slammer effectively implemented a denial-of-service on enterprise networks even for uninfected machines as the worm overwhelmed the capacity of many LAN segments. Policy enforcement of out-of-profile traffic at the origin mitigates this type of bandwidth DoS.

As we shall discuss in depth below, a primary conclusion of our analysis is that in order for a policy to have both reasonable security and usability, it must differentiate between communications that use fixed server port numbers and those that use somewhat random port numbers that are defined on the fly (e.g., agreed upon over the fixed-port communication). We refer to the latter as *ephemeral ports*. In principle, given the profile of allowed fixed-port communication, it is possible to dynamically populate and enforce rules for ephemeral-port communication. All such approaches, however, require an additional protocol and/or modifications to existing protocol software. While such approaches may eventually allow for fined-grained control of ephemeral ports, in the interest of deployability, in this work we considered only a restricted design space of policies that are enforceable in a host or a switch without the use of additional protocols or the modification of existing application software.

The remainder of the paper is structured as follows. Section 2 presents related work. Section 3 introduces the data set we used. In Section 4 we describe our approach of building end host profiles and describes the different throttling disciplines that we investigate. We then evaluate the usability and security of our policies in Section 5 and Section 6 respectively. The paper concludes in Section 7.

2 Related Work

Administrators in the early days of the Internet relied almost exclusively on network perimeter defenses. Perimeter defenses were primarily implemented through *firewalls* that physically govern the traffic entering and leaving the network. Managing the often low-level access controls on firewalls has shown to be quit complex and multiple efforts

have focused to ease this burden on the network operator [4, 1, 2, 10, 18].

Despite those efforts, perimeter defenses became less effective as the Internet grew. Enterprises began to employ the myriad of technologies and applications at their disposal, e.g., multi-media, B2B applications, remote data. These services required significantly more access to resources external to the enterprise. Hence, the perimeter was necessarily weakened as more channels to these services were opened. Bellovin addressed the dissolving perimeter by applying the firewall policies at each end-point [3, 9]. Centralized entities in his *distributed firewall* system distribute firewall policies to each host in the network. Because global policy is enforced through the aggregate behavior of host, the dissolving perimeter is less problematic. Personal firewalls (e.g., host-level firewalls) are now common features of almost all commercial and open source operating systems. There are several differences between [3, 9] and the work presented in this paper. We also envision enforcement of communication policies at, or close to, the end-host. However, our policies treat the end-host as potentially hostile and seek to protect the rest of the enterprise network by blocking illegitimate packets emanating from that end-host. The perspective of [3, 9] is to protect each end host from other potentially hostile hosts and is primarily concerned with blocking illegitimate packets at the destination. In addition, we consider the issues of brownfield deployment and out-of-profile throttling disciplines.

Our work is also related to more recently proposed, *introspective network systems*. Early systems such as GrIDS [15] used models of normal connectivity graphs to detect malicious behavior. These solutions were useful in detecting scanning attacks. As worm and virus behavior began to be better understood [14, 12], previous graph based mechanisms were replaced with more sophisticated behavioral models. For example, Staniford's CounterMalice [13] uses the network infrastructure to detect and counteract malicious activity. A model of normal activity is gleaned from historical data reflecting both number of unique IP addresses to which a connection is attempted and the number of failed connection attempts. This model of normal behavior is used to detect abnormal behavior that is then blocked. This *virus throttling* [17] can greatly reduce the infection rate of a worm. Our work differs from these earlier approaches in a number of ways. First is our use of communities of interest (COI) to derive the model of normal behavior. This builds on our earlier analysis of COI in enterprise networks [16] and is, to our knowledge, the first application of the COI concept to data networks. Second, is the fact that we use the model of normal behavior not to detect abnormal behavior, but to control the extent to which abnormal communication is allowed. This is a subtle but significant difference because there is no "detection phase" and therefore hosts are protected as soon as the security policy is in place.

3 Experimental Data Collection and Preprocessing

To perform the analysis presented in this paper, we collected eleven weeks worth of flow records from a single site in a large enterprise environment consisting of more than 400 distributed sites connected by a private IP backbone and serving a user population in excess of 50,000 users. The flow records were collected from a number of LAN switches using the Gigascope network monitor [5]. The LAN switches and Gigascope were configured to monitor *all* traffic for more than 300 hosts that included desktop machines, notebooks and lab servers. A 150 host subset of these 300 machines communicated during the entire eleven-week period. We refer to this smaller set as the *local hosts* and they form the focal point of our analysis. In addition to some communication amongst themselves, the local hosts mostly communicated with other hosts in the enterprise network (referred to as internal hosts) as well as with hosts outside the enterprise (i.e., external hosts). The latter communication is excluded from our analysis. During the eleven-week period we collected flow records corresponding to more than 4.5 TByte of network traffic. In this study we only considered TCP and UDP traffic. We also removed weekend data from our data set, thus ensuring a more consistent per-day traffic mix. Our measurement infrastructure generated "raw" flow records that were processed using the same techniques described in [16]. In summary this processing removed all unwanted (abnormal) traffic from the data, dealt with DHCP issues, and tagged data with client/server designations based on whether a host was initiating communication or listening on a socket to allow other hosts to communicate with it.

4 Securing the End Host

Our approach is to devise an individual policy for each end host based on the historical set of other hosts in the enterprise with which it interacts, i.e., its Community of Interest (COI). We perform this task in two steps. First, we develop a *COI Profile* of each end host. The goal of the *profile* is to capture what communication is normal for a particular host. Due to the cleaning that we perform on our data, we make the reasonable assumption that the data set we use for our analysis contains only legitimate traffic. In the second step we define *Throttling Disciplines (TDs)* that define what should happen if a host attempts to communicate outside of its profile. For each host, the combination of its profile and TD is used to restrict and control future

communication. We focus in particular on policies that can be easily implemented as access control lists (ACLs) on the switch that connects the host to the rest of the network. It should be immediately apparent that restricting communication in this way would have desirable security properties, because communication outside the "normal" profile can be restricted. At the same time, however, communication is never static and the system should minimize disruptions of legitimate traffic that falls outside of the historical profiles. Indeed, this tension between the usability of our approach and its security benefits forms the basis for our evaluation in Sections 5 and 6.

4.1 COI Profiles

In our approach, all *COI profiles* are constructed by analyzing historic network communication for hosts within an enterprise network during a profile training period. The goal of a COI profile is twofold. First it captures the historic communication of a set of end hosts, and second, this history is used to predict the host's future communication behavior.

Profiles differ in terms of the domain knowledge used to construct them, i.e., protocol specific information. The most basic of such profiles that we denote as *Simple COI Profiles* do not consider any domain specific knowledge, rather it only considers information from the 4-tuple: (protocol, client IP, server port, server IP). [1] Not surprisingly a Simple COI Profile that considers port level information will have great difficulty predicting the host's future communication behavior in the presence of applications that use ephemeral-port communication such as FTP and most streaming protocols. To compensate for the presence of ephemeral-port communication we investigate two alternate approaches. In the first approach we still consider only *Simple COI Profiles* but use wild cards for some of the fields in the four-tuple. As we will see this approach leads to profiles with unacceptable security properties. To address this we also investigate a heuristic that compensates for ephemeral communication while still considering the server port within the profile that we call the *Extended COI Profile*.

4.1.1 Simple COI Profiles

For a given set of clients we define pure history based profiles at the following three levels of granularity:

1. **Protocol, Client, Server Port, Server Profile (PCSPP)** - The PCSPP profile contains all (protocol, client IP, server port, server IP) tuples of historic communications for the given set of clients. This profile most closely represents past communication.

2. **Protocol, Client, Server Profile (PCSP)** – The PCSP profile contains all the (protocol, client IP, server IP address) tuples for the given set of clients. In other words, in this case, if a particular client communicated with a particular server on *any* port, we consider communication on all ports between the client and server to be part of the profile.

3. **Protocol, Server Profile (PSP)** – The PSP profile contains all (protocol, server IP address) tuples for the given set of clients. For this profile, if *any* client communicated with a server, then communication between *all* clients and this server (on all ports) is considered part of the profile.

4.1.2 Extended COI Profile

The observation that historical communication on ephemeral ports is not a very good predictor of port numbers used for future ephemeral communication is key to deriving an Extended COI profile. Intuitively there are three parts to our approach. First, we partition the training data into ephemeral and non-ephemeral communication. Second, the non-ephemeral training data is used directly to derive the first part of the Extended COI. Third, for the ephemeral training data, we assume that past communication is a good predictor of the future *occurrence* of ephemeral communication between the hosts in question. However, such future communication might not be on the observed server ports, and therefore we derive a more relaxed (inclusive) profile for the Extended COI profile.

A critical part of this approach is the accurate partitioning of the training data. Rather than relying on various manually derived heuristics (e.g., thresholding or application heuristics), we opted to use an automated data clustering approach. Rather than attempting to identify ephemeral communications directly, we go through two rounds of identifying (and removing from the training data), *non-ephemeral* communication. For each transport protocol, i.e., TCP and UDP, the following steps are executed:

1. **Non-ephemeral Global:** In the first step the algorithm considers the set of server ports and identifies those ports that are heavy hitters in terms of the *number of associated connections that are used by large numbers of servers*. Intuitively, this identifies the ports associated with popular fixed-port services that are running on multiple servers. We model this as a multidimensional clustering problem where the 2 dimensions are (i) number of connections per port and (ii) number of servers using that port. Each port can be represented as

[1] We refrain from considering client port for two reasons. First, client port numbers are not typically meaningful and therefore will have no bearing on the client profile. Second, as we will show in Section 5, a profile that considers server ports is already overly restrictive thus hampering its usability.

a point in this 2-dimensional space. We then use the *k-means* unsupervised statistical clustering algorithm [7] to separate the heavy hitter ports from the others. Given a set of points in an n-dimensional space, and a desired number of partitions k, *k-means* performs an iterative partitioning which minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. Our inputs to the k-means algorithm are all ports that were used as server ports on multiple servers. Each port p is represented by the tuple (n_p, s_p) where n_p is the number of connections on that server port and s_p the number of servers that used port p. We set k to 2 to separate the heavy hitter cluster, and used squared Euclidean distance as the distance measure for clustering.

Before applying k-means clustering we perform the following operations. (i) *Log transformation*: in order to reduce the effect of outliers at the high end of the value range, transform the data values for each variable to a logarithmic scale. (ii) *Scale Standardization*: The 2 variables have different scales. We normalize them on a common scale to avoid one variable from dominating the other in the clustering. We selected the widely used z-score normalization. Z-Score is defined as $x = (v - mean(v))/stdev(v)$, where v is the input variable, and the transformed scores have a mean of zero and a standard deviation of one.

The k-means algorithm uses randomly selected initial centroid locations. Poorly placed centroids may drive the k-means clustering to a local minimum. To address this possibility, we repeat the k-means clustering 100 times, each with a new set of randomly selected initial cluster centroid positions, and select the solution with the lowest value for the total sum of within-cluster point-to-centroid distances. The k-means step yields 2 distinct clusterings of the points. The first one corresponds to points clustered around low values of number of connections and number of servers. The second cluster consists of points that have high values along these dimensions. The ports corresponding to these points are selected as the *global ports* for the transport protocol $prot$ being considered. We derive the following profile for these ports. The tuple (c, s, p, $prot$) is added to the profile if there was a client c communicating with a server s on global server port p with protocol $prot$.

2. **Non-ephemeral Per-Server:** We remove from the training data all the communications associated with the global ports identified above. In the next step we aim to identify from among the remaining ports, those (server, port) pairs that have significant communications. Intuitively, each such pair corresponds to that server running non-ephemeral communications over that server port. We use the k-means clustering (described above), with $k = 2$ for clustering based on *number of connections* from each (server, server port) pair. We select the heavy hitter cluster as the set of *per-server ports*. The tuple (c, s, p, $prot$) is added to the profile if there was a client c communication with a server s on server port p with protocol $prot$, where the pair (s,p) belongs to the heavy hitter (server, server port) set. For each server s, we also compute the list of ports P_s, which will be used in the next step, that are either in the global port list or in the heavy hitter port list.

3. **Ephemeral:** The next step is to identify, from the remaining communications those client-server pairs that communicate on many ports, i.e., client-server pairs engaging in ephemeral communication. We use k-means clustering to cluster the client-server pairs into a heavy-hitter and non-heavy-hitter cluster based on the *number of ports* these pairs communicate on. For the identified heavy user pairs (c, s), we add rules to the profile that would allow communication between the pair on **all ports**, except the set of global ports or the per-server heavy hitter ports (set P_s) for that server. This exclusion is geared specifically to protect the ports in P_s which are running popular fixed-port services.

4. **Non-Ephemeral Unclassified:** In the final step all remaining communications that have not been classified through the preceding steps are added to the profile as non-ephemeral communication. I.e., if there was a client c communicating with a server s on server port p with protocol $prot$, then we add the tuple (c, s, p, $prot$) to the profile.

4.2 Throttling Disciplines

We consider the following Throttling Disciplines (TDs): (i) n-r-**Strict:** A n-r-strict TD blocks all out-of-profile communication and blocks all communication of a client if the number of out-of-profile attempts exceed a threshold n within a time period r. n might be 0 in which case all communication is blocked as soon as one out-of-profile communication is attempted. (ii) n-r-**Relaxed:** A n-r-relaxed TD allows n out-of-profile communications to succeed within a time period r. If the number of out-of-profile interactions exceed a threshold n within a time period r, all future communication is blocked. For $n = 0$ this policy is equal to the 0-strict policy. (iii) n-r-**Open:** A n-r-open TD allows n out-of-profile communications within a time period r. It blocks all out-of-profile communications when the number

of out-of-profile interactions exceed a threshold n within a time period r, but never blocks in-profile communications.

5 Usability

In this section we discuss the usability of the profiles and throttling disciplines we proposed. In particular we evaluate three aspects of usability. First we discuss the different profile sizes that will impact the complexity required to implement such a profile on a network device. Secondly we investigate how well profiles built using the first four weeks of our data set match communication in the next four weeks to provide some insight in how well profiles predict future communication. In the last evaluation of this section we simulate the three TDs.

5.1 Profile Properties

Since our profiles need to be specified within the network device (switch/router/firewall) that enforces our TDs, the size of each profile is of particular importance. Using the profiles we derived from the first four weeks, we computed the number of rules needed to specify the profile. In this context a rule has slightly different definitions for the different profiles. For simple history based profiles, a rule is defined as the n-tuple covering the fields included in the profile. This means that the PCSPP rules are defined as the following 4-tuple (c, s, p, $prot$). The PCSP rules are a 3-tuple covering (c, s, $prot$) and the PSP rules are a 2-tuple including (s, $prot$) . In contrast, the Extended COI Profile has two types of rules. The first type of rules covers all non-ephemeral communication and is represented by a 4-tuple identical to the PCSPP rules. The second type of rule describes ephemeral communication. These rules are represented by a tuple representing a client, a server and a set of port ranges. We denote these rules as range rules.

As expected the rule counts for the three simple COI profiles (PSP, PCSP, PCSPP), shown in Table 1, show an increasing number of rules as more IP header fields are added to the profile. These profiles show a 4.2 to 6.3 fold increase in rules going from a purely server IP address and protocol based rule set to a rule set based on client IP address, server IP address and protocol. Interestingly, the further increase in profile rules if the server port is added is 59% for TCP whereas for UDP the profile size increases 374%. This result confirms the intuition that TCP server ports are more stable than UDP server ports.

The number of rules needed to specify the Extended COI profile is only slightly larger than the number of rules in the PCSPP profile. In both the UDP and TCP cases we required less than 400 ephemeral rules for our client set of 150 hosts. Overall we can conclude that our profile sizes are quite manageable. Even for the largest profile (Extended COI) we require on average less than 100 rules per client to capture the clients communication pattern.

Next we consider the question of how well the history-based profiles predict future behavior of clients. Figure 1 shows the empirical CDF of connections per client in weeks 5 to 8 of our data sets. In the rest of this section we will use weeks 5 to 8 as test weeks to validate the profiles we derived from the data collected in weeks 1 to 4. Therefore, a basic understanding of the traffic in weeks 5 to 8 is of interest. First we can observe that we see similar distributions in all 4 weeks for the empirical CDF of the number of client connections. This allows us to conclude that each of the 4 test weeks has a comparable mix of client traffic. In addition we observe that more than 45% of the clients have more than 1000 connections within the enterprise network per work week, a reasonable model for normal user behavior.

Using these test weeks, we counted the number of connections not within the PCSPP profile. Figure 2 depicts the empirical CDF of such connections. The number of missed connections is fairly high. For example, 20% of the clients miss at least 100 connections per week in each of the weeks. Although this number is not outrageously high, it is still fairly significant. Note that the miss count is really a conservative estimate of the true user impact and does not account for collateral impact. For example, the 0-r-Strict TD will block all the out-of-profile connections, as well as all in-profile communications during a blocking event. This highlights the need for a policy that allows for some level of out-of-profile communications to accommodate normal changes in communication patterns.

5.2 User Impact

We estimate the impact on a user for a given profile via simulation of all combinations of profiles and policies under various parameter settings. The goal of the simulator is to determine how many legal connections would have been blocked in the simulated environment. The simulator uses an out-of-profile counter c that counts the number of out-of-profile connections. It is either reset after the time period r specified in the TD or manually by the network operator following an investigation of the *event*. As our data has been cleaned, in this section we assume that all events are caused by "good" traffic that happens to be out-of-profile. An *event* occurs when the out-of-profile counter c exceeds the threshold n of allowed out-of-profile communications before c is being reset. Therefore, the simulator considers the following parameters:

- **Profile:** The derived profiles namely, PSP, PCSP, PCSPP, and extended COI.
- **TD:** The throttling policies namely, STRICT, RELAXED, and OPEN.
- n**:** The allowed out-of-profile threshold. We investigated values of 0, 1, 5, 10, 15, and 20.

Algorithm	Protocol	n-tuple rules	range rules	Total
PSP	TCP	705	0	705
PCSP	TCP	3001	0	3001
PCSPP	TCP	4775	0	4775
Extended COI	TCP	4443	338	4781
PSP	UDP	316	0	316
PCSP	UDP	2014	0	2014
PCSPP	UDP	9563	0	9563
Extended COI	UDP	9193	389	9582

Table 1. Profile Sizes

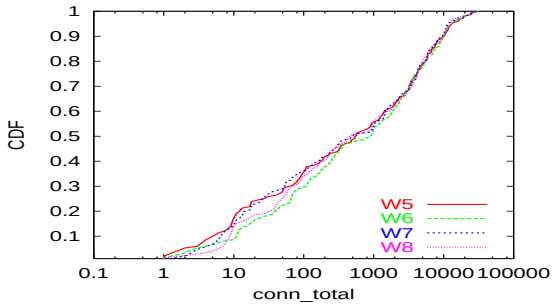


Figure 1. Total Connections per Client

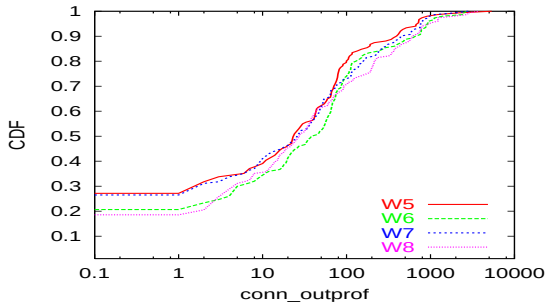


Figure 2. Missed Connections per client in PCSPP

- r**:** The counter-reset-timer r is the time interval[2] in which we reset c to zero. We investigated 1 hour and 1 day.
- **Block Time:** The time period it takes an operator to investigate an event and unblock a client. After a client is unblocked c is set to zero. We investigated values of 1 minute, 10 minutes and 1 hour.
- **Test Week:** The week for which we performed the simulation. We used all test weeks which are weeks 5-8.

The simulator measures the number of blocked connections and blocking events.

5.2.1 Simple COI Profile Based Simulation Results

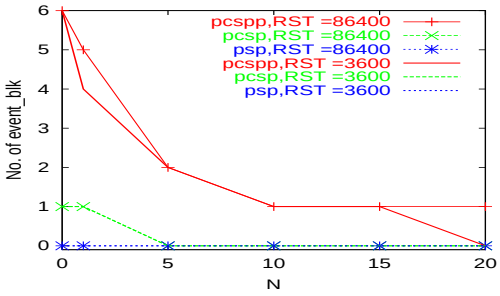
We performed simulations for all Simple COI based profiles under the various parameter settings. To reduce the parameter space, we then grouped all results for a given profile, TD, n, block time, and r set into a single representing the numberof blocked connections per (client, week) tuple. Since the test weeks have similar empirical distributions, as shown in Figures 1 and 2, we believe that this combination has minimal impact on the results presented while simplifying the presentation of the results in this multidimensional parameter space.

Blocked Events: Figure 3 depicts the number of events for the 50%-tile and 90%-tile of (client, week) tuples. These two figure covers all three TDs since the number of events is independent of the chosen TD. The reason for this independence is that an event is triggered when the TD threshold n is reached, which represents the start of an event in all three TDs. TD-specific blocking action is taken for the duration of the block time, but does not come into play if we are only considering event counts. The out-of-profile count, c, is reset either after an event or after the reset time expires. The times in this cycle do not depend on the TD used, therefore, the event count is independent of the TD.

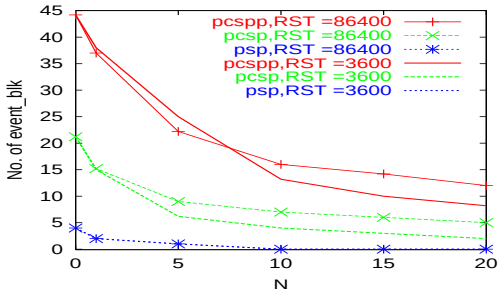
In Figure 3 the block time was fixed to 10 minutes to investigate a blocking event. As expected the graphs show that the number of events decreases as n increases. In fact for $n = 10$ the 50%tile graph shows only 1 event per (client, week) pair for the most restrictive Simple COI based profile. The 90%tile event graph shows that the event count increases by more than an order of magnitude.

Blocked Connections: In Figures 4, 5, and 6, we show the 50 and 90%tile results of the number of blocked connections for OPEN, RELAXED and STRICT TDs respectively. It is not surprising that the OPEN TD performs best in both the 50%tile and 90%tile. This is because the in profile communication is never blocked. However, as we will see in the security analysis (Section 6) this TD allows all vulnerable in-profile servers to be infected by a spreading worm. In contrast the RELAXED and STRICT TD block all communications after an event which has a modest impact on the 50%tile number of connections blocked per (client, week) but a dramatic impact on the 90%tile number of connections

[2]An alternate approach to resetting c after a fixed interval would be to use a sliding window of size r. We chose the interval-based approach since it lends itself to a simpler implementation

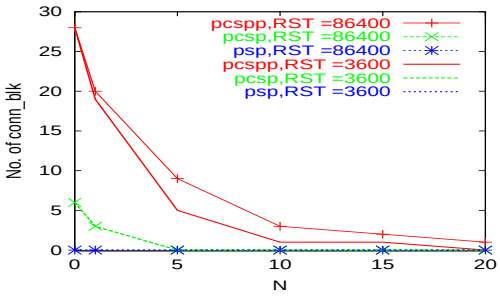


(a) 50%tile

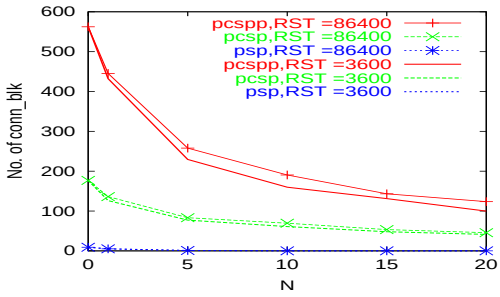


(b) 90%tile

Figure 3. Number of Blocking Events using 10 minute block time



(a) 50%tile



(b) 90%tile

Figure 4. Blocked Connections for OPEN policy using 10 minute block time

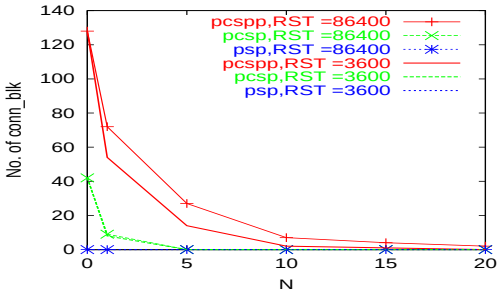
blocked per (client, week). The reason is for this is that if a busy client triggers an event, many in profile communications suffer. In addition the STRICT TD always blocks out of profile communications even if no event occurs. Compared to the RELAXED TD, this slightly increases the number of blocked connections per (client, week) for small n (in fact the two TDs are equal for $n = 0$) but shows a larger impact for large n.

It is not surprising that the Simple COI based profiles are becoming substantially less usable as additional IP header fields are considered. For example, for the RELAXED TD in Figure 5, the number of blocked connection in the 90%tile graph differs by three orders of magnitude between PSP and PCSPP for $n = 1$. Although this would indicate that from a usability point of view we should choose one of the less restrictive profiles, we will see in Section 6 that the security properties of the PSP and PCSP profiles are not acceptable for a large number of environments.

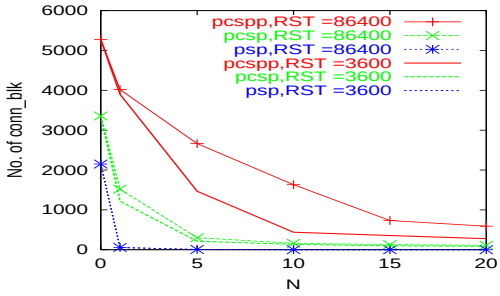
Another observation we can make from Figures 3 to 6 is that r, which we varied between 1 hour (RST=3600 seconds) and 1 day (RST=86400 seconds), does seem to impact the results sub-linearly. In nearly all cases a doubling in n improves the results approximately the same amount as the reductions of the r from 1 day to 1 hour. Since the amount of out of profile communication which is possible without triggering an event is limited by the rate $oprate = n/r$ it seems beneficial to chose a r of 1 day which is 24 times larger than the 1 hour reset time and compensate for the increase in events or blocked connections by doubling n. The resulting $oprate$ is therefore 12 times smaller while we preserve the same usability characteristics. The $oprate$ is particularly important for slow spreading worms which can only spread undetected (without triggering an event) if they initiate out of profile connections with a rate smaller than $oprate$.

Blocked Time: Based on the results of previous experiments, we chose $n = 10$, r =1 day and the RELAXED TD for the remaining evaluations. This choice was motivated by the fact that for the tested TDs and parameter settings a larger n seems to produce diminishing returns. In addition, as discussed above, choosing a r of 1 day greatly reduces the undetected probing a slow spreading worm can perform without impacting the number of events or blocked connections per (client, week) substantially. The relaxed TD was chosen since the OPEN TD, as we will see later, is too permissive from a security perspective while the STRICT TD produces a higher user impact.

Using these settings we now investigate the impact of the block time. This parameter is determined by how quickly network operators react and is not a parameter that can be chosen freely. The results, shown in Figure 7, indicate that the number of connections blocked per (client, week) increases sub-linearly with increasing block time. We observe that a block time of 10 minutes (which requires the network

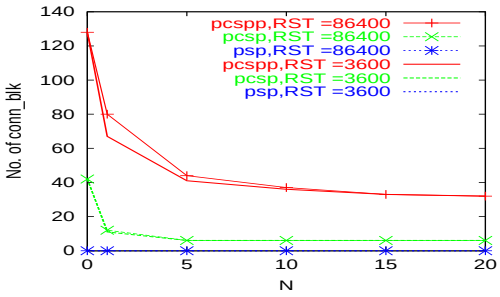


(a) 50%tile

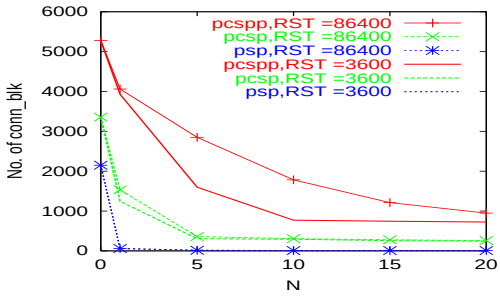


(b) 90%tile

Figure 5. Blocked Connections for RELAXED policy using 10 minute block time



(a) 50%tile



(b) 90%tile

Figure 6. Blocked Connections for STRICT policy using 10 minute block time

operator to react to events within that time) provides acceptable results with 7 blocked connections per (client, week) for the 50%tile PCSPP profile.

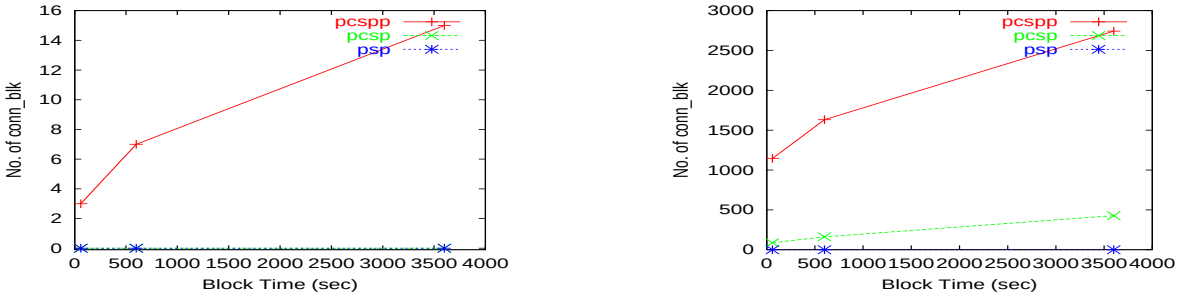
5.2.2 Impact of Extended COI

As we have discussed before, a substantial part of the out-of-profile connections in the PCSPP profile are due to ephemeral ports. Our goal with developing the Extended COI profile is to attempt to more accurately predict such ephemeral communication. We now investigate the benefits in terms of usability of the Extended COI profile by comparing it to the PCSPP profile with the parameter set derived in the previous section. Table 2 shows the absolute number of events and blocked connections for the 50, 80 and 90%tile (client, week) value. It also shows the relative *improvement* of the events and blocked connections for those percentiles. We focused this table on the 50-90% tile range since the lower percentiles already produce few events and blocked connections. The results show that the number of blocked connections improves between 43% to 66% for the 50%tile, between 30% to 39% for the 80th percentile, and between 15% to 33% for the 90th percentile—overall a substantial improvement over the PCSPP profile. The number of events is also reduced, however, it seems that the improvements are smaller which indicates that busier hosts benefit more.

6 Security

This section details our simulation-based security evaluation of the COI approach. We use the data and methodology identified in the preceding sections to derive a network model and source TD and profiles. The worm simulation used throughout models propagation in simulated (discrete) rounds within a modeled enterprise network. The simulated worm can compromise hosts through a fixed port over which the worm is propagated, e.g., port 80/`http`. An *infected* host can comprise exactly one other host in a round. Hence, modulo the mitigation technique used, the number of worms attempting to compromise other hosts grows at each round. In this way, the simulation models the exponential behavior of worm propagation within the enterprise. We use two metrics to evaluate security. The *infected* metric indicates the number of hosts infected at the end of the experiment. This reflects the degree to which the tested policies contain the worm outbreak. The second metric, *time to completion* measures the number of rounds till completion of the experiment. This measures how quickly a worm is contained or, in the case of saturation, how quickly all vulnerable hosts are compromised.

Each compromised host in the simulation has a fixed probability s of successfully compromising another simulated host in a round. This parameter models the ability of the worm to identify a vulnerable host to attempt to compro-



(a) 50%tile (b) 90%tile

Figure 7. Blocked Connections per (client, week) vs Block duration (RELAXED policy 1 day reset time and $n = 10$)

Comparison Type	PCSPP	Exten.	Change %	PCSPP	Exten.	Change %	PCSPP	Exten.	Change %
	50%			80%			90%		
Events	1	1	0	7	5	27	16	14	11
Open BC	3	1	66	71	42	39	190	160	15
Relaxed BC	7	3	57	266	160	39	1632	1090	33
Strict BC	37	21	43	451	313	30	1782	1340	24

Table 2. Extended COI and Simple COI based profiles events and blocked connections (BC)

mise. A host can be compromised if (a) the host is reachable by the infecting host according to the tested profile and TD, and (b) the policy has not restricted communication to the victim host/port, and (c) if the victim host is not already compromised. If a host is infected during one round of the simulation, then it will attempt to infect other hosts in subsequent rounds. Every out-of-profile communication attempt by a host is deemed a miss. As defined in Section 4, TD determines how the host is restricted once it reaches the n threshold of misses. To restate the TDs, the n-r-STRICT and n-r-RELAXED will prevent any further compromise attempts by that host once the threshold is reached, and the n-r-OPEN will prevent any out-of-profile attempts once the threshold is reached. For our simulation we assumed that r is much larger then the time to spread the worm. For brevity we, therefore, do not consider the impact of r, i.e., assume $r = \infty$.

An experiment terminates when all hosts are compromised or there are no hosts that can compromise any remaining uninfected hosts. The network is *saturated* if all compromisable hosts are compromised (which represents the worst-case, total infection of the network). Conversely, a worm is *contained* when the worm can no longer spread because of profile and TD restrictions, but there remain hosts that are vulnerable to compromise. The simulation uses the source data used in the profile creation to seed the experiments. We conservatively assume all hosts that have the target port in their profile, either as a client or a server port, are vulnerable. In practice, there may be other ports vulnerable in the network that were unused over the 8 week sampling period. On first glance, this may appear to underestimate the possible spread of the worm. However, because unused ports will not show up in any profile, they offer no additional opportunity for comprise in a n-r-STRICT policy, and only up to the threshold in the n-r-RELAXED and n-r-OPEN policies. Conversely, many worms are only able to compromise hosts that are listening on a particular port, i.e., will only exploit a vulnerability in the server interface. Our simulation of client ports as also being vulnerable is more conservative, as it would require the compromising of the client based on client request, e.g., a buffer overflow in the response to some client driven request. In practice, client-side vulnerabilities are less frequently observed and are more difficult to exploit. Hence, the propagation of worms in real networks is likely to be less successful than that reported.

Each experiment seeds a single simulated host as compromised and models infection behavior, round by round, until the experiment terminates. We repeat the simulation for each seeded host with a different seed and show the measurements taken over all experiments. Hence, each experiment is for a TD/profile/parameter is repeated a number of times equal to the number of vulnerable hosts. For our evaluation we chose ports for which a reasonably large number of exploits is actively being monitored by a large tier 1 ISP e.g., 25, 80, 53, 135, 137, 139, 443, 445. We repeat the simulation over the three TDs (n-r-STRICT, n-r-RELAXED, and n-r-OPEN) and four profiles (PC-SPP, PCSP, PSP, and Extended COI). Table 3 summarizes the number of infectable hosts tested ports. Note that by con-

Port	COI Profiles			
	PC-SPP	PCSP	PSP	Extended
25/tcp	40	135	135	40
80/tcp	91	135	135	91
53/udp	128	151	151	128
135/tcp	54	135	135	54
137/udp	119	151	151	119
137/tcp	0	135	135	61
139/tcp	98	135	135	98
443/tcp	69	135	135	69
445/tcp	90	135	135	90

Table 3. Vulnerable hosts by protocol

struction, all hosts will be modeled as vulnerable in the PCSP and PSP policies because they do not restrict communication based on port.

6.1 Baseline Infection Rates

Figure 8 depicts the analytically developed spread of a worm in a network with no worm mitigation in place. This represents the worst-case scenario where all hosts are vulnerable and there is no countermeasure in place to detect and mitigate the worm behavior. Capped at 10,000 host infections, these tests show how the success of the worm in finding other victims affects the length of time it takes to saturate the network. The curves demonstrate why worms are so dangerous—even for worms with very small infection rates (1 in 10,000 attempts leads to an infection), there exists a point of critically at which the outbreak is growing at a rate faster than can reasonably be controlled. Highly effective worms (for example 100%, such as those with hitlists [14]) essentially infect the entire network immediately, where it takes only 14 rounds to infect the entire network. The slope of the infections is slightly less steep in the worms with lower rate. However, they are still extremely dangerous: using Slammer's average rate of 7 infections a second [11], it would take slightly less than 4 seconds for a worm to infect a network of 10,000 hosts.

Studied hereafter, there are two goals of a countermeasure illustrated by these graphs. First, obviously, it is desirable to completely contain the outbreak. Stopping an infection before it saturates the local network, and possibly more importantly, before it becomes uncontrollable within the Internet is essential to quashing it. However, as described by Staniford et al., it is not always possible to stop the worm. In this case, slowing the rate of infection is the primary goal: decreasing the slope of infection is enormously useful because it allows other countermeasures to be enacted. Such other countermeasures span from simple (turning off vulnerable hosts) to the exceptionally complex (real-time patching [8]) countermeasures.

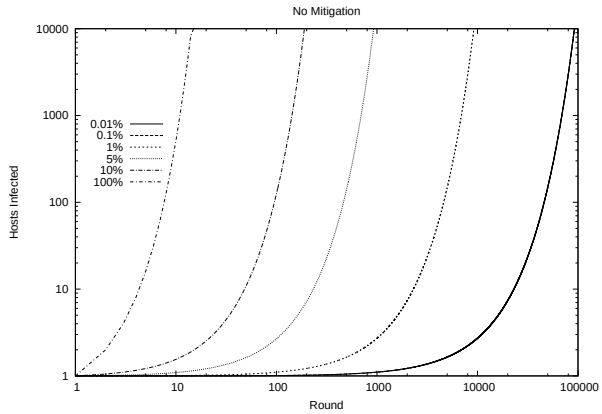


Figure 8. Unabated worm propagation

6.2 Worm Containment

An initial set of experiments sought to measure the effectiveness of the STRICT and RELAXED TD. Figure 9 shows a CCDF of the results of tests that simulate a worm attacking port 137 with $s = 0.01(1\%)$ and $n = 10$ under STRICT and RELAXED TDs. This aggressive worm successfully compromises a host one out of every one hundred tries but is largely ineffective. It is quashed in short order because any infected host is shut down before it can do much damage: that is after ten misses, the host is prevented from communicating over the network. The STRICT policies almost never goes beyond a single host – no out of profile communication is ever allowed. However, the RELAXED policy allows a small amount of additional spread (by allowing some infections to occur that are out of profile).

The simulation of worm behavior under the OPEN TD lead to more polar results. Either the worm was largely contained as in the preceding experiments (infecting < 5 hosts) or all hosts were compromised. Figure 10 illustrates this phenomenon for the same experimental setup as above, e.g., port 137, $s = 0.01(1\%)$, and $n = 10$ for the OPEN policy. Note that the profile types begin to exhibit different levels of effectiveness at this point. The PSP policy is essentially ineffective against the worm: over 98% of the tests ended with 140 of the 151 hosts being infected and over 50% of the tests ended with all the hosts being infected. Almost all networks protected by a PSP profile are saturated, save a vanishingly small number of cases where the mechanisms prevent infection of a few largely inactive hosts. Similarly, in the PCSP profile, the worm's behavior was largely divided, where 47% of the tests resulted in 3 or fewer infections and 46% of the tests resulted 139 or more infections. The Extended COI and PCSPP profiles performed essentially the same, where the contained/full infection rates occurred at around 30%. This similarity is a reflection of the profile construction, which, for this port, is identical. Note that the

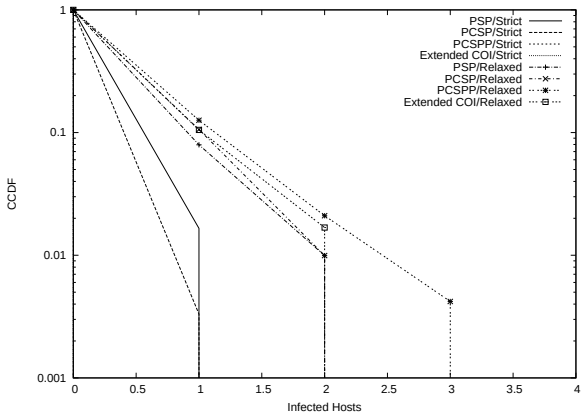


Figure 9. Worm infections on port 137, s=0.01**,** $n = 10$

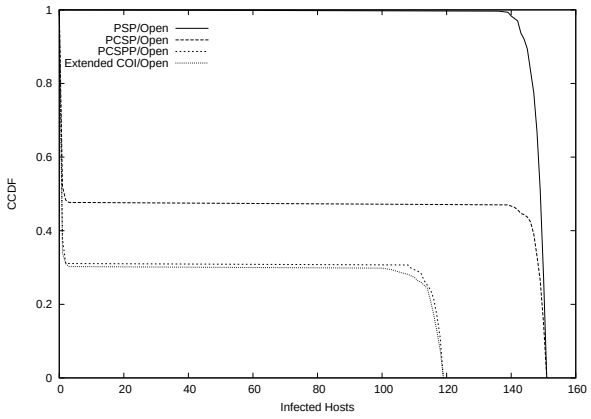


Figure 10. Worm infections on port 137, s=0.01**,** $n = 10$

polar results are not surprising. The containment/total infection split essentially places a probabilistic roadblock in worm spread. Based on the worm aggressiveness and effectiveness, one can select a profile and TD that simply prevents it from reaching the point of criticality. Moreover, the Extended COI and PCSPP policies are significantly more effective in preventing the spread of the worm.

6.3 Worm Moderation

A key question asks how long the worm can continue to survive in a restricted network. Each network reaches a point at which the worm has compromised as many hosts as is possible under the enforced profiles and TDs. We use the same set of experiments to characterize the worm *lifetime*, e.g., port 137, $s = 0.01(1\%)$, and $n = 10$. A worm lifetime is measured in either the amount of time it continues to infect hosts (where containment is achieved) or the length of time that is required to saturate the network. Interestingly, we desire to reduce the lifetime in the former case, e.g., to contain the worm as quickly as possible, and lengthen the lifetime in the latter, i.e., to allow for the worm to be recognized and other countermeasures employed.

Figure 11 shows the lifetime of the worm in the simulated environments under the STRICT and RELAXED policies. The lifetimes reflect the effective containment illustrated in the preceding section. Almost all experiments end in 10 rounds. The 10 round lower bound occurs when the worm stays alive while it consumes its $n = 10$ out-of-profile grace connections, after which it is completely shut down and thus contained. Note that a compromised host will find another host that is infectable by *somebody* with probability 0.1. In the Extended COI, PCSP, and PCSPP profile, just because the host is infectable does not mean that the host is infectable by the client attempting the infection. Hence, because these otherwise infectable hosts can count as misses for those clients, few hosts survive more than the 10 rounds. Conversely, the PSP profile does not restrict client behavior, and hence we see many more outbreaks—as expected, around 10% of the hosts compromise one or more hosts under the PSP profile.

The ability of our approach to slow worm propagation is illustrated in Figure 12. As in the preceding section, the OPEN policy leads to polar behavior. Either the worm is contained quickly or the network is saturated after a long period. The interesting aspect of this graph is not the containment, but the saturation. The expected baseline infection rates state that, under the experimental parameters, the worm should saturate the network in 482 rounds for Extended COI and PCSPP, and 506 rounds in the PSP and PCSP profiles. Our experiments show that this actual time to saturation is significantly longer, where the worm needs up to four times longer to saturate the network. Depending on the network size and available detection and defensive apparatus, this may allow sufficient time to enact effective countermeasures.

We performed the experiments detailed in the preceding sections for all the protected ports. Summarized in table 4, the results were largely similar to those in the preceding experiments. All ports exhibited polar results: either all hosts were compromised or very few were. For example, the appendix shows that the diverse ports 53 (DNS) and 80 (HTTP) exhibit the similar protective characteristics as port 137 (NETBIOS) in the preceding section.

Our experiments showed how the parameter s influences the worm success. We found that highly effective ($s > 10$) worms are largely impossible to stop under RELAXED and OPEN TDs. However, even a worm with perfect accuracy (e.g., a flash worm) is only able to infect one third of the hosts in a PSP profile, and essentially no hosts using the Ex-

Port	Policy	s(%)	n	PSP	PCSP	PCSPP	Intelligent
135/tcp	strict	1	10	0.768%	0.741%	1.852%	1.852%
135/tcp	strict	5	10	0.872%	0.741%	1.852%	1.852%
135/tcp	strict	5	100	14.044%	0.785%	1.852%	1.852%
135/tcp	strict	10	100	31.048%	0.818%	1.852%	1.852%
135/tcp	strict	25	1000	33.421%	10.126%	1.852%	1.852%
135/tcp	strict	100	1000	33.421%	12.617%	1.852%	1.852%
135/tcp	relaxed	1	10	0.842%	0.793%	2.143%	2.109%
135/tcp	relaxed	5	10	1.383%	1.495%	3.841%	3.738%
135/tcp	relaxed	5	100	98.938%	98.996%	99.280%	99.331%
135/tcp	relaxed	10	100	99.997%	99.995%	100.000%	100.000%
135/tcp	relaxed	25	1000	100.000%	100.000%	100.000%	100.000%
135/tcp	relaxed	100	1000	100.000%	100.000%	100.000%	100.000%
135/tcp	open	1	10	92.060%	61.871%	1.989%	1.972%
135/tcp	open	5	10	95.734%	50.209%	16.907%	10.065%
135/tcp	open	5	10	98.621%	98.886%	99.949%	99.074%
135/tcp	open	10	100	100.000%	100.000%	99.983%	100.000%
135/tcp	open	25	1000	100.000%	100.000%	100.000%	100.000%
135/tcp	open	100	1000	100.000%	100.000%	100.000%	100.000%
53/udp	strict	1	10	0.673%	0.662%	0.781%	0.781%
53/udp	strict	10	10	0.969%	0.667%	0.781%	0.781%
53/udp	relaxed	1	10	0.739%	0.732%	0.861%	0.854%
53/udp	relaxed	10	10	6.125%	6.456%	6.796%	6.110%
53/udp	open	1	10	98.719%	46.456%	0.897%	0.858%
53/udp	open	10	10	99.737%	75.157%	54.831%	55.530%
80/tcp	strict	1	10	0.763%	0.741%	1.099%	1.099%
80/tcp	strict	10	10	1.152%	0.752%	1.099%	1.099%
80/tcp	relaxed	1	10	0.815%	0.815%	1.177%	1.220%
80/tcp	relaxed	10	10	6.505%	6.549%	7.457%	7.946%
80/tcp	open	1	10	91.907%	50.486%	25.999%	26.186%
80/tcp	open	10	10	96.524%	78.875%	62.764%	67.534%
137/tcp	open	1	10	91.536%	51.254%	**N/A**	7.041%
137/tcp	open	10	10	96.840%	79.597%	**N/A**	64.888%

Table 4. Broad security experiments - number of hosts infected per policy on the tested ports

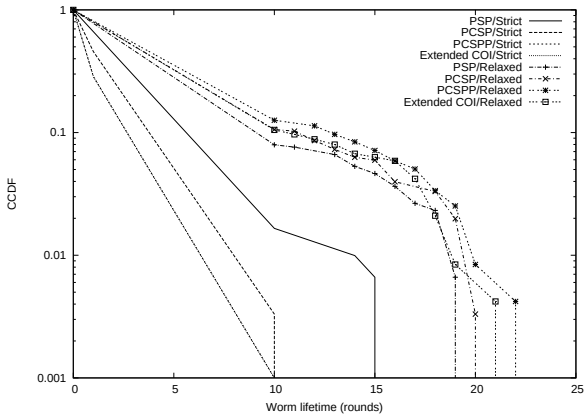


Figure 11. Worm lifetimes on port 137, s=0.01**,** $n = 10$

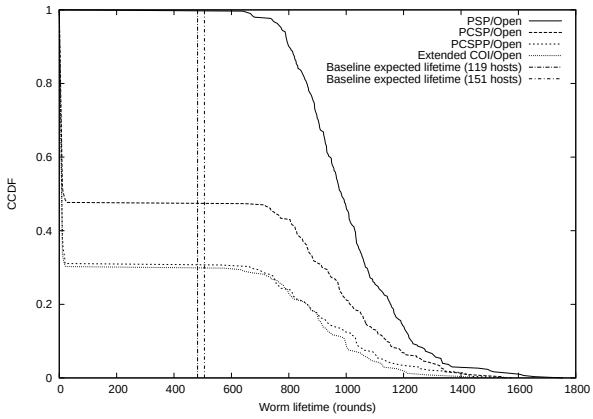


Figure 12. Worm lifetimes on port 137, s=0.01**,** $n = 10$

tended COI. The reason for this is that the there are many hosts that may be infectable, but only a minute few that are actually within a particular Extended COI profile. Altering the policy threshold n similarly affects the result: there are demonstrable result changes in almost all policies when changing the n parameter from 10 to 100. As the threshold increases, the worm has more opportunity to infect out-of-profile hosts. Hence, as is seen on port 135 in the RELAXED TD, increasing n allows a containable worm to saturate the network.

7 Conclusions and Future Work

In this paper we presented a brownfield approach to hardening an enterprise network against internally spreading malware. We presented techniques for automatically deriving four different individual host profiles (PSP, PCSP, PCSPP and extended COI) to capture historical communication patterns (i.e., community of interest (COI)) of end hosts within an enterprise network. Using traces from a large enterprise network, we explore how three different security policies based on these profiles and TDs impact both the usability of the end host as well as the spread of worms and similar security threats. The main conclusions are: (i) The results validate a key premise of our approach that if future communication patterns are constrained to historical "normal" communication patterns, then the ability of malware (e.g., worms) to exploit vulnerabilities in the enterprise can indeed be severely curtailed. (ii) While the overall approach is promising, the selection of the correct profile and throttling discipline are both crucial to achieving effective security and usability properties. PSP and PCSP possess good usability but were too permissive resulting in poor security impact, while PCSPP had good security but poor usability, due to its inability to accommodate ephemeral communications. Among the throttling disciplines, $n - r - Open$ had good usability but was largely ineffective against worm attacks, while $n - r - Strict$ was very effective in worm defense but lacked good usability. (iii) Allowing a small number ($n = 10$) of out-of-profile communications per host before the throttling action is initiated, can substantially improve the usability (by accommodating new valid communications), while maintaining good security properties of the policy. (iv) A simple security policy comprised of our *Extended COI profile* and *n-r-Relaxed Throttling Discipline* which allows a small number of out-of-profile communications per host, achieves both high protection against worms and other threats, as well as low impact on the usability of end hosts.

We have shown that our profiles are sufficiently stable for weeks at a time, but still need to address how these profiles will be updated as communication patterns change over longer time periods. One possible option is to recompute the profiles periodically including the not-blocked out-of-profile communications in the recomputation process. An alternate approach is to have the network operator add rules for legitimate out of profile communications manually whenever the network operator investigates an event. We are currently investigating the tradeoffs of these and other options.

8 Acknowledgements

We would like to thank Haowen Chan for his many contributions to early work in automated policy generation, to Will Enck, Lisa Johansen, and Patrick Verkaik for their editorial comments, and to the anonymous reviewers for their insightful technical comments.

References

[1] K. Al-Tawil and I. A. Al-Kaltham. Evaluation and testing of internet firewalls. *International Journal of Network Management*, 9(3):135–149, 1999.

[2] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31, 1999.

[3] S. Bellovin. Distributed Firewalls. *;login:*, pages 39–47, Nov. 1999.

[4] W. Cheswick and S. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, 1994.

[5] Chuck Cranor et. al. Gigascope: a stream database for network applications. In *Proceedings of ACM SIGMOD*, June 2003.

[6] Cooperative Association for Internet Data Analysis (CAIDA). The Spread of the Witty Worm, March 2004. `http://www.caida.org/analysis/security/witty/`.

[7] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[8] M. Hicks. *Dynamic Software Updating*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, August 2001. Winner of the 2002 ACM SIGPLAN Doctoral Dissertation award.

[9] S. Ioannidis, A. D. Keromytis, S. Bellovin, and J. M. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 190–199, Nov. 2000. Athens, Greece.

[10] A. Mayer, A. Wool, and E. Ziskind. Fang: A Firewall Analysis Engine. In *2000 IEEE Symposium on Security and Privacy*, pages 177–187. IEEE, May 2000. Oakland, CA.

[11] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. W. ver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.

[12] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of IEEE INFOCOM 2003*. IEEE, March 2003. San Francisco, CA.

[13] S. Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, 2004. *to appear*.

[14] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167. USENIX Association, 2002.

[15] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS A Graph-Based Intrusion Detection System for Large Networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[16] W. Aiello, C. Kalmanek, P. McDaniel, S. Sen, O. Spatscheck and J. Van der Merwe. Analysis of Communities of Interest in Data Networks, March 2005. Passive and Active Measurement Workshop.

[17] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 61–68. IEEE Computer Society, 2002.

[18] A. Wool. Architecting the Lumeta Firewall Analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001. Washington, DC.