A Distributed Host-based Worm Detection System

Senthilkumar G. Cheetancheri[1], John Mark Agosta[2], Denver H. Dash[2],
Karl N. Levitt[1], Jeff Rowe[1], Eve M. Schooler[2]

[1]UC Davis
Dept. of Computer Science
Davis, CA - 95616. USA
{cheetanc, levitt, rowe}@cs.ucdavis.edu

[2]Intel Research
2200 Mission College Blvd.
Santa Clara, CA 95052. USA
firstname.MI.lastname@intel.com

ABSTRACT

We present a method for detecting large-scale worm attacks using only end-host detectors. These detectors propagate and aggregate alerts to cooperating partners to detect large-scale distributed attacks in progress. The properties of the host-based detectors may in fact be relatively poor in isolation but when taken collectively result in a high-quality distributed worm detector. We implement a cooperative alert sharing protocol coupled with distributed sequential hypothesis testing to generate global alarms about distributed attacks. We evaluate the system's response in the presence of a variety of false alarm conditions and in the presence of an Internet worm attack. Our evaluation is conducted with agents on the Emulab and DETER emulated testbeds using real operating systems and computing platforms.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software*; G.3 [**Probability and Statistics**]: *Sequential Hypothesis Testing*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Sequential Hypothesis Testing*

General Terms

Security

Keywords

Worm detection, Sequential Hypothesis Testing

1. INTRODUCTION

Monitoring for and responding to security incidents in large-scale, complex enterprise networks requires a new approach to security incident management. Security reports indicating a policy violation, come from a heterogeneous collection of components, such as intrusion detection sensors, firewall access policy violations or anomalous network traffic loads. Protecting against attacks currently in progress or eliminating a new vulnerability involves the reconfiguration of several different types of devices, such as firewalls, border gateways, software updates, and even host-based wrappers.

The challenge is to collect all information from the numerous data sources and to decide on appropriate actions for each reactive component. Simply forwarding all reports to a central location will not scale to large networks. Local decision making, however, may lack the global view necessary to thwart large-scale attacks.

Defending against worms, particularly day-zero worms, is perhaps the most pressing challenge for a large enterprise. Such a worm can have a devastating impact as it automatically propagates itself to all vulnerable machines on a network. Defending against worm attacks, for which no pre-existing attack signature is available, requires the automation of tasks that current system administrators must perform manually. These include: automatic aggregation and correlation of security reports to detect activity at a local site, automated short-term defensive actions to stop local worm infections, cooperative alert sharing across administrative boundaries to protect sites not yet infected, and automated back-off when a worm is contained or in the event of a false alarm.

2. A DISTRIBUTED COLLABORATIVE DEFENSE

As a complement to centralized cyber-security defensive systems we have developed and evaluated cooperative defensive schemes. Centralized systems are designed primarily to protect enterprises by monitoring aggregate traffic at fixed locations in the network and responding by blocking or delaying observed malicious behavior. In some circumstances, however, such centralized systems may not be suitable; organizations may not have the resources to acquire and manage a large system, there may not be sufficient trust between sub-domains to accept a centralized protection policy, and large numbers of mobile nodes may exit and enter the network leaving them temporarily without protection.

Previous work by us and others[11, 2, 5, 6] have developed cyber-defenses based upon collaborative alert-sharing as a way to detect and react to large-scale distributed attack such as Internet worms. Evaluation of these schemes is usually done both analytically and through simulation. Assumptions regarding the false positive rates are idealized

SIGCOMM'06 Workshops September 11-15, 2006, Pisa, Italy.

abstractions due to the lack of a realistic testing and evaluation framework.

2.1 Collaborative Distributed Attack Detection

In this paper we describe and evaluate a scheme for distributed attack detection using cooperating end-hosts. In this system, all events are generated using software detection agents on individual end-hosts. Currently, we monitor inbound and outbound network traffic at the host and detect local anomalies in traffic features. Due to the limited view of these detectors, however, isolated end-hosts alone would serve only as low-quality (high false positive or high false negative) detectors of distributed attacks. Our goal is to cooperatively share information such that the aggregation of end-host alerts produces a high-quality (low false positive and low false negative) global attack detector. We accomplish this by implementing a distributed version of the sequential hypothesis test(SHT) used successfully in centralized detection schemes[12]. With this method, all collaborating sites maintain a decision table constructed using the ratio of the likelihood that the features are a good indicator of the current worm attack to the likelihood for the features to occur at random. When the observed behavior exceeds a predetermined threshold, enough evidence has been accumulated to reach a correct decision with high probability.

In this formulation, let H_1 and H_0 be the hypotheses that there is and is not a worm respectively. Let Y_i be the random variable that says there is an attack or not at site i. This represents the weak local end-host detector at site i.

$$Y_i = \begin{cases} 1 & \text{if there is an attack;} \\ & \text{could be a false positive(fp)} \\ 0 & \text{if there is no attack;} \\ & \text{could be a false negative(fn)} \end{cases}$$

By definition,

$$\mathrm{P}[Y_i = 0|H_1] = \mathrm{fn}; \quad \mathrm{P}[Y_i = 1|H_1] = (1 - \mathrm{fn})$$
$$\mathrm{P}[Y_i = 1|H_0] = \mathrm{fp}; \quad \mathrm{P}[Y_i = 0|H_0] = (1 - \mathrm{fp})$$

The observation vector $\vec{Y} = \{Y_1, Y_2 \cdots Y_n\}$ then is the set of measurements obtained by n conditionally independent end-hosts. We then define the Likelihood Ratio from the observation as,

$$\mathrm{L}(\vec{Y}) = \frac{\mathrm{P}[\vec{Y}|H_1]}{P[\vec{Y}|H_0]},$$

and assuming all Y_i's are independent measurements, we have,

$$\mathrm{L}(\vec{Y}) = \frac{P[Y_1|H_1].P[Y_2|H_1].\cdots.P[Y_n|H_1]}{P[Y_1|H_0].P[Y_2|H_0].\cdots.P[Y_n|H_0]}$$

for a sequence of n local detectors sampled. Then if vector $\vec{Y}$ has a 1's and b 0's, the Likelihood ratio is,

$$\mathrm{L}(\vec{Y}) = \frac{(1 - \mathrm{fn})^a * \mathrm{fn}^b}{\mathrm{fp}^a * (1 - \mathrm{fp})^b}$$

Using this we compute a table of the outcomes of many random walks through a collection of local detectors. For example, entry (5,2) would contain the likelihood ratio of finding two alerts after sampling five independent sites.

The strength of the desired global detector, then, is specified by two quantities: desired detection rate, DD, and desired false alarm rate, DF. DF, in other words, is the maximum acceptable failure rate of the global detector. Using these, one can calculate thresholds in the table of likelihood ratios:

$$T0 = \frac{1 - DD}{1 - DF} \quad \text{and} \quad T1 = \frac{DD}{DF}$$

Each host, then, implements a global intrusion detector that make decisions as follows: if, after including the local detector state, the calculated likelihood ratio, $\mathrm{L}(\vec{Y}) < T0$, accept the hypothesis that there is no worm (H_0) and halt the query. If $\mathrm{L}(\vec{Y}) > T1$, accept the worm hypothesis (H_1) and raise a global alarm, otherwise continue the random walk among end hosts. This defines upper and lower blocks in the decision table as a region likely to have been produced by an attack and a region likely to come from normal behavior. By independently sampling weak local end-host detectors with given *fp* and *fn*, one can achieve a strong global detector if enough sites are traversed.

2.2 Cooperative Messaging Protocols

In the scheme described above, the method for obtaining random samples from cooperating end-hosts is left unspecified. In the case of Internet worm attack, our initial tests were performed using an epidemic spread protocol. Cooperating hosts contain a random subset of the addresses of all nodes in the collection. Nodes with new alerts from their local detectors choose m other end-hosts at random and send the message "{1, 1}", which means, "one site has reported one alert". Hosts receiving this message add their local information (e.g. it would generate a "{2, 1}" if had not seen the activity, and a "{2, 2}" if had) and attempt to arrive at a decision based upon the table of likelihood ratios. If no decision is reached, m new sites are selected at random and the message propagates. In this manner multiple SHT sequences(chains) of evidence are spread randomly across cooperating end-hosts. If "normal behavior" decisions are reached in any chain, that chain halts. If a "likely worm attack" decision is reached at any point, a global warning is broadcast to all nodes. Figure 1 shows an example message chain with a fan-out, $m = 2$. Preliminary experiments on Emulab[23] and DETER[4] testbeds have led us to conclude that messaging overheads for protocols with $m > 1$ provide little benefit in early detection and result in needless communications in the presence of local false positives. During times of widespread attacks multiple query chains are initiated by local detectors, forming an ever-increasing number of independent queries.

3. EXPERIMENTAL EVALUATION ON AN EMULATED TEST-BED

One major difficulty in testing any large-scale defensive systems is that a large number of test machines have to be configured and managed efficiently. To accomplish these tasks, we have developed a preliminary worm defense testing framework[8] that wraps around existing network testbeds like Emulab and DETER.

This testing framework allows experimenters to rapidly deploy and easily repeat large-scale worm experiments using several hundreds to thousands of machines. These experiments can be used to analyze the efficiency of novel defenses against different kinds of worms.

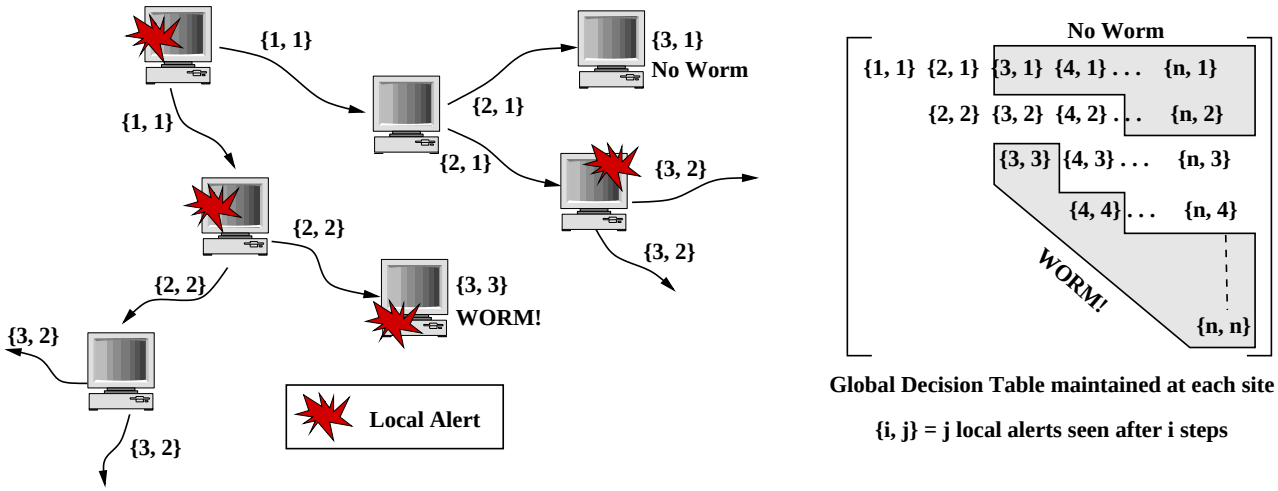

Figure 1: Diagram illustrating the co-operative messaging protocol and the decision table used in SHT used to generate a global worm detector

This framework receives a description of our network topology format along with the detection engine and compiles them in a "NS-2" format required by the testbed. Though it provides its own library of worms and a server that is vulnerable to those worms, we override these two components with a more powerful worm simulator engine called *Wormsim*[17] and its companion XML worm-specification library. These components are described in detail in the next section.

The Event Control System(ECS) is the central software piece in the Emulab testbed that helps run experiments and collects logs. Our framework wraps the ECS system. It does this in the following order. The detection engine and *Wormsim* are started on all nodes. Each *Wormsim* agent is then configured to be either vulnerable or not to a particular worm on a certain port. A random process chooses which nodes are vulnerable based on the vulnerability density specified along with the user network topology. Alternatively, specific nodes can be set to be vulnerable in the topology specification. The experiment is started by launching a seed worm to one of the vulnerable nodes.

The experiment's progress is monitored by observing the logs recorded by the detection engine. The experiment is deemed complete once a 'worm' decision is recorded in the log, or when no more logs are recorded for a pre-determined period of time. Once an experiment is completed, the ECS collects logs from the various experiment nodes in a database and initiates an experiment with new parameters. An overall architecture for our worm testing framework is shown in Figure 2.

3.1 Experimental Setup

The goals of our experiments are to evaluate our algorithms' effectiveness in identifying worm outbreaks, to determine its robustness against false alerts and to measure the network overhead of the cooperative protocol itself.

The major components of our current experiment setup are:

- A worm simulator engine(*Wormsim*)
- A local intrusion detection system (IDS) to generate low level sensor inputs.

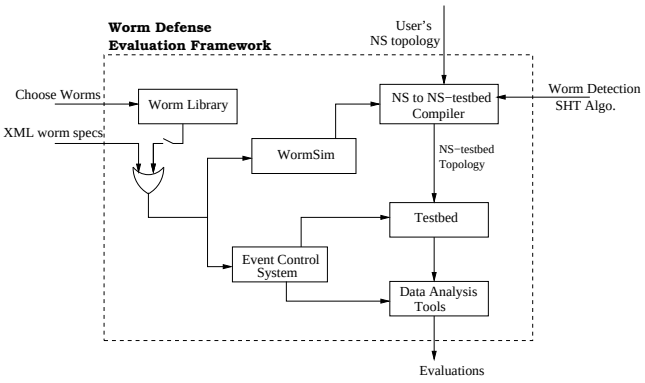

Figure 2: A worm defense evaluation framework architecture

- A global detection algorithm and protocol implementing the distributed sequential hypothesis test.
- The evaluation infrastructure including the network test-bed itself and instrumentation toolkits.

We describe these components briefly.

Wormsim. To test distributed defenses in the presence of realistic worm attacks without installing vulnerable software, we developed the Wormsim worm emulation framework. The goal of this framework is to generate network traffic patterns that mimic, as closely as possible, the patterns generated if malicious code had actually existed on the end hosts. Rather than executing malicious binary instructions that govern worm propagation, *Wormsim* agents interpret XML specifications written to emulate the same behavior. Agents accept and parse messages in an XML format and then, based upon the specification, connect to other "victim" hosts, sending them the same XML worm instructions. The targets are identified based upon the parameters in the XML worm specifications. Some other features that can be specified are the scan method, the transport protocol to use, the scan rate, etc. Each *Wormsim* engine can be re-

motely configured to be either vulnerable or not vulnerable to a particular worm.

A Local IDS. In tune with our philosophy of achieving high-confidence correlations from weak detectors, we implemented a very weak end-node IDS. The IDS would raise an alarm to trigger SHT whenever there is a connection attempt to an un-serviced port. The reasoning is that, a legitimate connection attempt usually never goes to a host that does not service it. On the contrary, automated attacks such as worms, ignoring sophisticated ones that have information from prior reconnaissance, try to make connections indiscriminately. Thus, those nodes that service a certain port have no protection and do not trigger the SHT. However, IDSes used in practice are much better than the one we use and will help in detecting a more sophisticated attack. This will enable even the vulnerable nodes to participate in the protocol.

Since *Wormsim* knows the vulnerability status of the host at a certain port, it can easily use the event of receiving XML specifications on a non-vulnerable node to trigger the detection algorithm. Hence this IDS was implemented as a patch to *Wormsim* itself.

Global Detection Algorithm. The sequential hypothesis test (SHT) detection algorithm and cooperative messaging protocol were implemented as a 'C' program. As a message propagates, each detection agent adds one to the number of nodes queried and one to the number of positives if it has seen a similar alert locally. At this time, we assume there is only one alert that can be raised and hence no information about the kind of attack is passed along. However, we envision using an anomaly vector in future to describe the event so that stronger correlations can be made.

The SHT parameters *DD* and *DF* were set to 98% and 2% respectively(section 2.1). The local IDS miss rate, *fn*, was set at 1%. Their false alarm rates were set as described in the next section. The fan-out, m, for the co-operative alert protocol (section 2.2) is set to 1. With each new infection, one new SHT sequence is created. After the first few infections, there are multiple parallel global alert chains propagating simultaneously. Each member propagates an alert by sharing it with another randomly chosen member. Besides satisfying a basic requirement for the SHT algorithm, such random selection defends the protocol from the following two attacks. One, malicious members gaming the protocol. Two, clever or hit-list worms, or a combination of both, circumventing our protocol by targeting only those nodes that will not be chosen to share the alert.

Evaluation Infrastructure. The experimental test network was configured with 100 PCs, a mixture of Pentium IVs and 64-bit Xeons randomly assigned by the testbed, running FreeBSD 4.10. All nodes were assigned to a single LAN, though we emphasize that we could have used several thousand machines. Each one of them can be as far away from each other on the Internet and only connectivity amongst the nodes matter. We also emphasize that we are not trying to save the entire Internet from the worm attack. We are only interested in an early detection for this particular federation of willing participants.

A 1Mb LAN was used so that test machines on different switches could be assigned to our experiment. This speeds up node assignment on the testbed to our experiment without significant changes in experimental results since our cooperative protocol was not expected to consume much of the total bandwidth.

4. EXPERIMENTAL RESULTS

To evaluate our system we focused upon three primary properties: the ability of the algorithm to detect worms, the likelihood of generating a global worm alert for a given level of local false alarms, and the messaging overhead of the system under various false alarm conditions.

4.1 False Alarm Experiments

Since, the local host IDS operates on a very naïve principle, we expect to initiate cooperative chains conducting the SHT quite frequently and on false pretexts. Each and every local false alarm, or even a malicious port scan, will initiate a query sequence. We take this into account by assigning to each local IDS a certain false alarm rate *fp*. That is to say, for each IDS, a certain n alarms out of every 100 will be spurious. To test the effects of the quality of the local detector on the global decision, we set the local host IDS quality at 5 different levels, where $n = (1, 3, 5, 10, 20)$. This property of the IDS forms an input to calculating the likelihood ratios that go into the decision table held by each participant. We perform experiments with one of these IDS quality settings at a time.

It would be impractical to use the false alarm rates, configured as a parameter of the local detectors, to generate sensor events in the test-bed experiment. Most of the experiment time would be spent in simply waiting for a rare event. Alternatively, we selectively generate the rare events themselves and record the effects of these events on our SHT algorithm. The goal here is to generate simultaneous false-alarm conditions so that a SHT sequence has multiple members that have seen a local false alarm. We use the Event Control System (ECS) of the emulab test-bed to trigger false alarms in a number, m, of participants simultaneously. We choose $m = (3, 5, 10, 20)$ in order in the experiments described below. The results, then, can be applied to systems with differing local host IDS false alarm rates using the likelihood of seeing 3, 5, 10 or 20 simultaneous false alarms for the given IDS.

Thus, we have a family of 20 experiments with different configurations (m simultaneous false alarm conditions times n local IDS quality levels) to run. We repeat each experiment 20 times to reduce the effects of random fluctuations. These experiments were conducted with the detection system running on all 100 nodes. There was no need for *Wormsim* to be run, except the IDS portion.

The first question we attempt to answer is; for a given number of simultaneous false alarms what is the chance that the distributed system will generate a false global worm alert? Figure 3 shows the fraction of times out of 20 repetitions of the false alarm experiments that the distributed SHT claimed that there was indeed a worm. Naturally the likelihood of false worm claims goes up as the number of simultaneous false alarms increases. However, as the quality of the end-node IDS goes down, the quality of the global detector goes up. For example, for a very poor quality local host IDS (with a 20% *fp*) the distributed SHT algorithm makes the global detector highly suspicious of alerts received resulting in fewer wrong decisions.

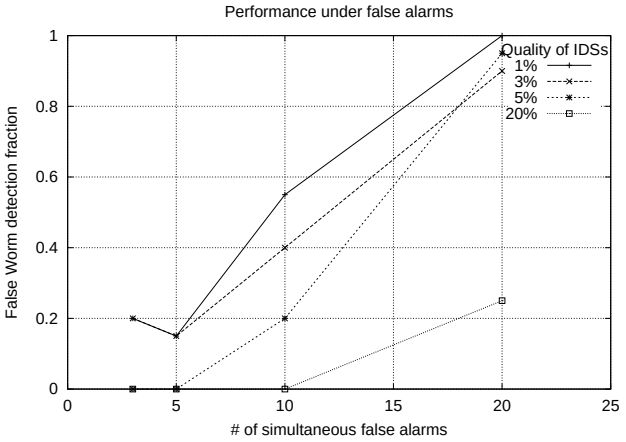

Figure 3: Wrong worm decisions due to false alerts

For the higher quality local host IDS, 5 simultaneous false alarms will produce a global worm alert using our distributed SHT 15% of the time. While this may not seem particularly small, the chance of getting 5 simultaneous false alarms to begin with will be quite small for these types of detectors.

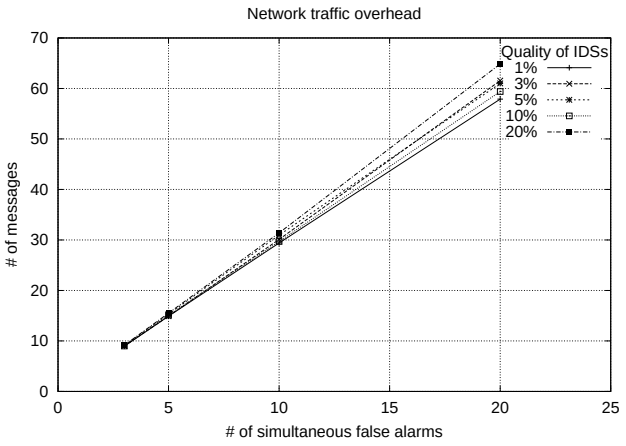

Figure 4: Total number of messages required before distributed SHT reaches a decision

The second question we wish to address is, how much network resources will be consumed by running this cooperative alert protocol under normal operating conditions? The concern here is that if the local host IDS quality is too low, during normal operations, the distributed SHT would require an excessive number of queries in each chain before a decision were obtained one way or the other. In essence, the path taken in the decision table would remain in the middle, undecided portion rather than reaching any decision, correct or incorrect. Were this to happen continuously it might adversely affect network operations or allow a sophisticated attacker to trigger minor false alarms to deliberately induce periods of high bandwidth message passing. Figure 4 shows the number of messages required to arrive at a decision, right or wrong, for the four levels of simultaneous false alarms and for five different qualities of the local end-host detectors. The numbers are averaged over 20 experiments. The maximum standard deviation observed was 3.2 messages; when 20 false alarms were fired simultaneously from end-node IDSes whose *fp* rates were pegged at 3%.

The number of messages increases in proportion to the number of simultaneous false alarms since each false alarm initiates a new query chain. The number of messages depends little, however, on the quality of the local end-host IDS. During periods of false alarms, since the local alerts are independently distributed across end-hosts (next hop neighbors are selected at random), decisions are reached regarding false alarms after querying only four end-hosts on average. There seems to be little danger here in a runaway distributed SHT algorithm causing harm to normal network operations, even when the local end-host detectors are relatively poor.

4.2 Performance in Detection Worm Attacks

The second set of experiments was performed to test the system's response in the presence of self-propagating worm attack. We do not study the effects of false alarms in presence of worm traffic as it would only help to make a "worm" decision sooner. For our worm experiments we set the vulnerability density to be 25%; a random process chooses which specific nodes in the test-bed are vulnerable. We configured the worm to send out a random subnet scan every 1 second. Since the entire vulnerable population is on one subnet, this worm is effectively a random scanning worm. The worm scan speed does not have any impact on detection unless it is faster than the detection algorithm. The morphology of the worm is also not of concern as we do not deal with worm semantics. We only exchange much coarser information about anomalies.

We want to determine the effect of various local end-host IDS quality on decision time and infection rates. Thus, we have n experiments to run against this worm; one for each end-host IDS quality. We again repeat this experiment 20 times to reduce the effects of random fluctuations.

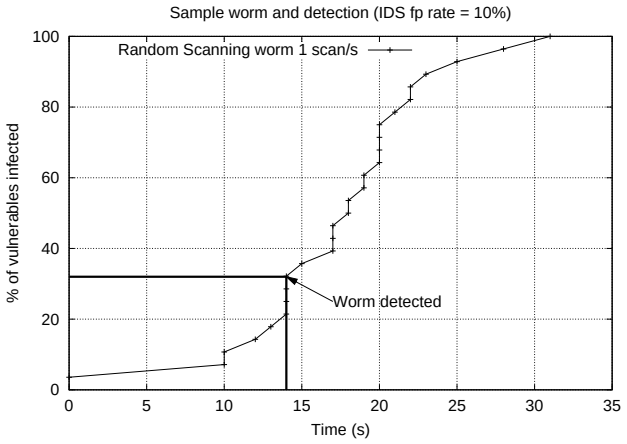

Figure 5: A sample infection sequence and detection instant

The results from a typical worm attack experiment are shown in Figure 5. The percentage of vulnerable machines infected is plotted as a function of time and exhibits the characteristic s-curve infection profile. In this example, the decision table is constructed using a 10% false alarm rate in the end-node detectors. At this rate, a worm decision is

reached at 14 seconds after the launch of the attack with 32% of the vulnerable nodes already infected. Since the local end-host IDS in this case is rather poor, a decision is not reached until relatively late in the infection profile.

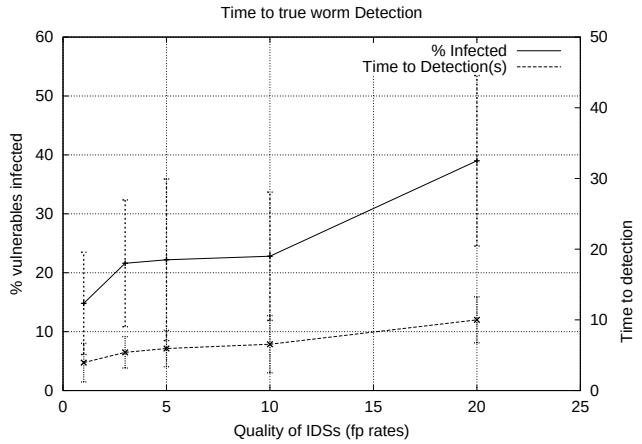

Figure 6: Results from all worm experiments showing the percentage of infected nodes at detection time as a function of local end-host IDS quality

Detection times and percentages of infected hosts from all experiments were collected and are shown plotted together in Figure 6. We notice that the number of infections before worm-detection increases with decreasing quality of end-node detectors. While poorer quality end-host detectors do not necessarily lead to larger problems with respect to false alarms, they have a significant impact on the global distributed SHT detector's ability to quickly detect worms; before unacceptable numbers of vulnerable nodes have been compromised. Since the global distributed SHT must be more tolerant to high levels of false alarms, it takes longer to claim a worm with the required high levels of confidence. However, there are no cases of missed worms. Sooner or later, worms are always detected. Only those worms that carefully avoid all non-vulnerable nodes will not be detected. However, as mentioned earlier in section 3.1, IDSes used in practice are much better than the one we used and can detect more sophisticated attacks enabling even the vulnerable nodes to participate in the protocol.

5. RELATED WORK

This work was inspired by the Jung et al.'s algorithm for quick portscan detection[12]. While they use Sequential Hypothesis Testing to identify malicious port-scanners, we adapted the principle to detect worms.

There is a vast literature on novel approaches to worm detection and defense including those that use collaborative techniques. We briefly review a few representative ones, consider their strengths and weaknesses, compare them with our approach, and acknowledge a few others[3, 9, 16, 2]. Zou et al.[24] present an algorithm for early detection of worms using a network of monitors employing Kalman filters and an aggregator that digest the observations sent by them. Their model suffers from single point failures and demands that observations be immediately available to the aggregator even in presence of a worm. These shortcomings make it difficult for deployment in production environment whereas our approach is completely distributed and there is no single point failure.

Sidiroglou et al.[20] approach the problem with end-point solutions. They use sand-boxing techniques to automatically generate localized patches to prevent worms from infecting production systems. However, they leave identifying worms to other third party systems like honeypots and IDSes.

Autograph[13] proposes a distributed content-based payload partitioning method to identify worms and their signatures. They propose multicasting information about suspect port-scanners to all participants in the distributed detection. Earlybird[19] is a very promising approach toward identifying and generating signature for zero-day worms. It uses content prevalence and dispersion of participating addresses. Nevertheless, it needs to be installed at a high-visibility site where large amounts of network traffic can be monitored. Monitoring at the border may be infeasible for some sites. Both of the above use Rabin fingerprints to characterize the suspicious traffic.

Columbia University's[21] uses its predecessor PAYL[22] to profile normal data and flag any data that does not match this profile. It first uses ingress/egress correlation. If there is a suspicious anomaly, it then tries to correlate that with one another site. If there is a match, a worm is declared and the correlated string is used as a worm signature. But this minimalist correlation is fraught with high false positives.

Cai et al.[7]propose a collaborative worm containment technique. It needs to be deployed on edge-networks and requires high processing power and careful manual oversight owing to its high-visibility location on the network. Furthermore, their work is also supported by simulations only.

Dash et al.[10] extend collaborative anomaly detection to distributed belief passing among end-hosts with outgoing traffic local detectors. Both this and our work is distinguished from all of the other work described above in that we do not need a monitor at a high-visibility place like at the border gateway or at the DMZ. Both leverage relatively simple and weak IDSes on individual end-host computers and make high confidence distributed correlations using simple anomaly vectors. Distributed detection also avoids single points of failure. Dash et al. support their performance results by extensive discrete-event simulation experiments. Complementing theirs, we evaluated the system in an emulated test-bed environment and have demonstrated the efficacy of our system using real software components that run on real operating systems.

6. FUTURE WORK

There are quite a few important aspects that we still need to address. A couple of them are listed below.

We need to define the anomaly vector to share amongst the detection agents. Some of the features of this vector could be a flag to indicate presence of machine instructions in traffic to servers, the size of such instruction sequences, the frequency of such connection attempts, the recent CPU usage statistics, etc.

In our current study we have not taken into consideration the effect of the worm traffic from outside our network of interest. To address this, we are developing an Internet scale-down node. This node represents the Internet external to our network and would generate traffic into our emulation network based on the mathematical model of the worm

specification. We may be able to make use of the work done by Liljenstam et al.[15] for this. Evaluating distributed SHT in the presence of traffic from this Internet Scale-down node forms our next step in this direction.

We have also not considered the effects of malicious nodes in the federation in these experiments. To overcome such problems, we could introduce several variations in the protocol. For example, instead of declaring 'worm' immediately after the first such decision, we could wait until a certain number of unique participants make the same decision. It is worth noting that a similar problem has already been formulated and solved by the systems community as the Byzantine General's problem[14, 18, 1]. Those solutions might help alleviate this problem at the cost of using more network bandwidth and a delayed detection.

7. ACKNOWLEDGMENTS

We would like to thank the staff at the Emulab and DETER testbeds for their excellent support. This work was sponsored by a generous grant from Intel IT Research and NSF NRT grant 0335299. We would also like to thank Jaideep Chandrashekar and Denys Ma for their valuable tips and reviews of this work.

8. REFERENCES

[1] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber. Some constraints and tradeoffs in the design of network communications. In *SOSP '75: Proceedings of the fifth ACM Symposium on Operating Systems Principles*, pages 67–74. ACM Press, 1975.

[2] K. G. Anagnostakis et al. A cooperative immunization system for an untrusting internet. In *Proceedings of the 11th IEEE International Conference on Networks (ICON03)*, pages 403–408, October 2003.

[3] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, and A. D. Keromytis. Robust reactions to potential day-zero worms through cooperation and validation. In *Proceedings of the 9th Information Security Conference (ISC).*, 2006. To Appear.

[4] R. Bajcsy et al. Cyber defense technology networking and evaluation. *Commun. ACM*, 47(3):58–61, 2004.

[5] L. Briesemeister, P. Lincoln, and P. Porras. Epidemic profiles and defense of scale-free networks. In *Proceedings of the I ACM Workshop on Rapid Malcode (WORM03)*, pages 67–75, Oct. 2003.

[6] L. Briesemeister and P. Porras. Microscopic simulation of a group defense strategy. In *Proceedings of Workshop on Principles of Advanced and Distributed Simulation (PADS05)*, pages 254–261, June 2005.

[7] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen. Collaborative internet worm containment. *IEEE Security and Privacy*, 4(3):34–43, May 2005.

[8] S. G. Cheetancheri, D. Ma, T. Heberlien, and K. Levitt. Towards a framework for worm defense evaluation. In *Proceedings of the 1st Malware Workshop, (IPCCC06)*, Phoenix, Az, Apr. 2006.

[9] M. Costa et al. Vigilante: end-to-end containment of internet worms. In *SOSP '05: Proceedings of the twentieth ACM Symposium on Operating Systems Principles*, pages 133–147. ACM Press, 2005.

[10] D. Dash et al. When gossip is good: Distributed probabilistic inference for detection of slow network intrusions. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI06)*, 2006. To Appear.

[11] D.Nojiri, J.Rowe, and K.Levitt. "Cooperative Response Strategies for Large Sacle Attack Mitigation". In *Proceedings of the DARPA Information Survivability Conference and Exposition.* DISCEX, 2003.

[12] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[13] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the USENIX Security Symposium*, 2004.

[14] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[15] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. "Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing". In *Proceedings of the I ACM Workshop on Rapid Malcode (WORM03)*, Washington, DC, October 2003.

[16] D. J. Malan and M. D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proceedings of the III ACM Workshop on Rapid Malcode(WORM05)*, pages 72–80. ACM Press, 2005.

[17] J. McAlerney. "An Internet Worm Propagation Data Model". Master's thesis, University of California, Davis., Sept. 2004.

[18] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[19] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the Sixth ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, Dec. 2004.

[20] S.Sidiroglou and A. D. Keromytis. Countering network worms through automatic patch generation. *IEEE Security and Privacy*, 3(6):41 – 49, Nov. 2005.

[21] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection(RAID05)*, 2005.

[22] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection(RAID04)*, Sept 2004.

[23] B. White et al. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[24] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of the 10th ACM conference on Computer and communications security(CCS03)*, pages 190–199. ACM Press, 2003.