An Approach for Detecting Self-Propagating Email Using Anomaly Detection*

Ajay Gupta and R. Sekar

Department of Computer Science
Stony Brook University, Stony Brook, NY 11794.
E-mail: {ajay,sekar}@cs.sunysb.edu

Abstract. This paper develops a new approach for detecting self-propagating email viruses based on statistical anomaly detection. Our approach assumes that a key objective of an email virus attack is to eventually overwhelm mail servers and clients with a large volume of email traffic. Based on this assumption, the approach is designed to detect increases in traffic volume over what was observed during the training period. This paper describes our approach and the results of our simulation-based experiments in assessing the effectiveness of the approach in an intranet setting. Within the simulation setting, our results establish that the approach is effective in detecting attacks all of the time, with very few false alarms. In addition, attacks could be detected sufficiently early so that clean up efforts need to target only a fraction of the email clients in an intranet.

1 Introduction

Email viruses have become one of the major Internet security threats today. An email virus is a malicious program which hides in an email attachment, and becomes active when the attachment is opened. A principal goal of email virus attacks such as Melissa [1] is that of generating a large volume of email traffic over time, so that email servers and clients are eventually overwhelmed with this traffic, thus effectively disrupting the use of the email service. Future viruses may be more damaging, taking actions such as creating hidden back-doors on the infected machines that can be used to commandeer these machines in a subsequent coordinated attack.

Current approaches for dealing with email viruses rely on the use of anti-virus software at the desktops, network servers, mail exchange servers and at the gateways. Detection of email viruses is usually based on a signature-based approach, where the signature captures distinguishing features of a virus, such as a unique subject line or a unique sequence of bytes in its code. This approach is effective against known email viruses, but is ineffective against unknown (i.e., newly released) viruses. To overcome this drawback, techniques have been recently developed that focus on virus behavior rather than its representation. Such "behavior-blocking" approaches detect viruses by using signatures of behavior, such as fast generation of emails or self-replication.

Although behavior-blocking is more effective against unknown viruses, it can still be fooled by carefully designed viruses that propagate slowly, or replicate after a period. For instance, if system is set to block the behavior that an email attachment should not cause generation of more than k other email messages, a virus that generates only $k-1$ copies will go undetected. Similarly, an email attachment that causes time-delayed

* This research was supported in part by NSF under grant CCR-0098154 and the Defense Advanced Research Agency (DARPA) under contract number N66001-00-C-8022.

propagation may also go undetected. More generally, a virus can employ a combination of low propagation factor, high incubation period, and randomization to evade behavior-blocking approaches.

An alternative approach for detection is one that focuses on the ultimate effect of self-propagating email viruses: increase in email traffic. Simple adaptations on the part of the virus, such as reducing the propagation factor below a certain threshold, introducing time delays or other randomizations do not alter this ultimate effect. For this reason, our approach is based on detecting email viruses based on increases in the volume of email traffic generated.

Given the variations in email traffic from one site to another, and from one time to another, it is difficult for manual development of characterizations of excessive email traffic. An alternative approach is to use machine learning — the system is trained to learn characteristics of normal email traffic, and then detect significant increases. In the context of intrusion detection, such *anomaly detection* approaches have been associated with relatively high false-alarm rates, as well as a moderate rate of false negatives (i.e., missed attacks). In this paper, we develop and study an approach that appears to be capable of detecting attacks with very low false alarm rate, while still being able to detect attacks reasonably early.

This paper first presents our approach for anomaly-based detection of the self-propagating email viruses. It begins with an overview of our approach in Section 2. We have studied the performance of this approach using two complementary experiments, both based on simulation. The first experiment focuses on creating stealthy virus behaviors, but uses a simplistic user model. The second experiment strives for more realistic user models, as well as more accurate reproduction of the behaviors of different software components of the email system, but the virus models are not as stealthy or variable as the first experiment.

Section 3 describes our first experiment. Our experimental results show that viruses similar to the ones that are prevalent currently, can be detected early. This is because such viruses are very "noisy." For stealthier viruses that use a small replication factor, detection is still achieved fairly early in our simulation, when a minority of email clients are infected. For the most stealthy viruses that use a combination of low replication factor and delayed propagation, a majority of the network is infected by the time of detection. In all cases, detection is achieved before the time the email server experiences a high overload. Since our technique promises to provide low false alarm rates, there is a potential to launch automated responses at detection time so as to quarantine emails with attachments on the mail server[1]. At this point, a more careful investigation of the virus can be performed, followed by a cleanup phase (on the email clients) if a virus is indeed present. Note that early detection of the virus reduces the cleanup costs significantly, as only a fraction of the computers in an organization need to be cleaned up.

The second experiment, described in Section 4, used a more elaborate user model. Moreover, an actual email system was used so as to make the simulation results more realistic. The goal of the experiment, conducted as part of a DARPA-sponsored research

[1] Such a quarantine will be effective in arresting further spread of the virus, assuming that viruses can spread only through attachments.

program, was to study the effectiveness of automated response to check the spread of such viruses. A number of signature and behavior based detectors were used in combination with our anomaly detector. The signature and behavior based detectors were tuned for early detection, but this meant that the more stealthy viruses would not be caught by them. The anomaly detector was therefore tuned for delayed but certain detection. The detection delay was artificially increased so that the anomaly detector will not raise an alarm until it is certain that any responses based on other detectors have failed. For this reason, our primary effectiveness criteria in this experiment was detection, rather than early detection. Of the hundreds of experiments conducted in this set up, there were 7 cases where the virus was not checked by other detectors, and in each of these cases, our anomaly detector was able to detect the attack. These experimental results show that our approach is effective, subject to the accuracy of the simulation models used in the experiment. They also indicate that our approach can complement other "behavior-blocking" approaches, which are typically tuned for early detection but may be fooled by stealthy viruses.

Some of the key benefits of our approach are:

- *Accurate detection.* In our simulation-based experiments, our approach demonstrated near-zero false alarm rates with zero false negatives (i.e, 100% detection). The latter is possible because of the nature of self-propagating email, wherein the email traffic keeps increasing until it is detected.
- *Robust against polymorphic and stealthy viruses.* Our technique is unaffected by polymorphic viruses. It promises to reliably detect stealthy viruses that pose challenges to previously developed detection techniques, although the detection may be delayed.

A practical benefit of our approach is that it has a low runtime overhead. Moreover, its learning phase is robust enough to operate without expert supervision.

While the above results are promising, they are tempered by the fact that they are based exclusively on simulated behaviors of email users. The first experiment used a particularly simple model for user behaviors: each user was modeled as a Poisson process. The second experiment used a non-uniform model taking into account such factors as address books. User behavior was simulated using a 3-state ("reading email," "composing email," and "idle") Markov process that makes random transitions between states that is governed by a set of transition probabilities. Thus the user model was much more realistic in this experiment. Nevertheless, it is well known in the context of anomaly detection that real system behavior tends to exhibit more variability than what can be observed in a simulation. Thus, the results obtained using simulation experiments cannot be directly extrapolated to real operating environments. Our ongoing work aims to address this weakness by using simulation only for the purpose of modeling viruses; normal email traffic will be taken from actual mail server logs.

2 Overview of Approach

Our approach is based on *specification-based anomaly detection* [36], a technique that combines state-machine specifications of network protocols with statistical machine-learning. In this case, the protocol models the interaction between email clients in an organization with the email server of the same organization. These interactions are called

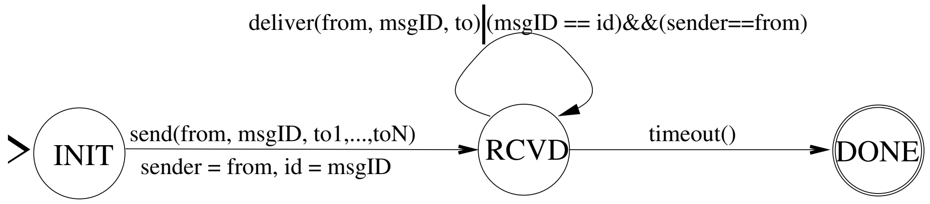

Fig. 1. A State Machine Modeling Email Server Operation

events. The state machine (implicitly) classifies events into different categories based on the transition taken by them in the state machine. Machine learning techniques are then used to learn statistics associated with each of these classes. Several choices exist for such statistics, including: average number of attachments to an email, size of a message, etc. Our focus, however, was on characteristics that are necessarily associated with increased email traffic, and hence we chose statistics relating to frequency of taking different transitions. The fact that this simple measure was effective supports the claim of [36] that the use of protocol state machines simplifies feature selection, i.e., even a naive choice of features produces good results.

The first step in our approach is to develop a state machine modeling the behavior of an email service, as observed at a mail server. For the rest of this paper, we concern ourselves mainly with email service within an intranet. We assume that all email clients transfer each of their outgoing messages to the intranet mail server, which in turn forwards the messages to each of the recipients[2]. Since we are only concerned with emails within the intranet, the email server simply queues each message received from any client on the mail queues associated with the respective recipients.

Figure 1 shows the simplified model of email server behavior described in the preceding paragraph. The state machine has three states that are identified as $INIT$, $RCVD$ and $DONE$. The reception of an email from a client $from$ at the server is modeled using the event $send$ that takes several parameters: the first parameter identifies the sender, the second is a unique identifier for the message, and the rest of the parameters denote the recipients of the message. The contents of the message are not modeled in this state machine. When the server receives this message, it forwards the message to each of the email recipients. This forwarding operation is modeled using the $deliver$ event, which takes the sender name, the message identifier and the recipient names as parameters. This event may occur zero or more times, depending on the number and email ids of the recipients. Note that there is no easy way to relate the number of recipients in the $send$ message with the number of recipients to which the message is forwarded by the server. The number of actual recipients of a message may be more (e.g., when a recipient name corresponds to a mailing list), or less (e.g., when a recipient name is in error, or due to duplicates or mail aliases within the recipient list). For this reason, the state machine indicates that there may be zero or more instances of the $deliver$ event corresponding to each $send$ event. The correspondence between the $send$ and $deliver$ events is identified in the state machine by storing the message

[2] This assumption holds for most popular email clients such as Microsoft Outlook and Netscape Messenger.

identifier and sender in two state variables id and $sender$, and then comparing these state variables with the arguments of the $deliver$ event.

The $DONE$ state in the state machine signifies the completion of the processing of a particular email from a client. Due to the difficulty outlined above in recognizing when such processing is completed, we use a time-out to model completion. The assumption being made here is that once an email is received by the server, it will be processed and the message sent to all recipients within a short period of time. The time-out value is set well above the expected time for such processing of email.

Formally, we use *extended finite state automata (EFSA)* to capture the state machine model shown in Figure 1. An EFSA is similar to a finite-state automaton in that it is characterized by a set of states, some times called *control states* of the automata, and a set of transitions between these states. EFSA differ from FSA in that (a) EFSA make transitions on events that may have arguments, and (b) EFSA can use a finite set of *state variables* in which values can be stored. The EFSA in Figure 1 consists of three control states $INIT$ (the start state), $RCVD$, and $DONE$ (the final state); three events $send$, $deliver$ and $timeout$; and two state variables $sender$ and id.

To understand how such EFSA specifications can be used for monitoring email traffic, consider the state machine diagram again. When an email is accepted by the mail server for delivery, a new instance of the state machine is created, and this instance makes a transition from $INIT$ to $RCVD$ state. The sender and message identifier are stored in the state variables associated with this instance. As copies of this message are delivered to the recipients, the $deliver$ transition is taken. Finally, after a timeout period, a transition to the $DONE$ state is taken. This being a final state, the state machine instance is no longer needed, and is cleaned up, i.e., all resources allocated for this state machine instance are released. Note that in general, there will be multiple instances of the state machine active at any time. The number of such active instances is determined by how many email messages are sent by clients over the duration of the timeout period.

Now, we superimpose statistical machine learning over this state-machine specification of email server behavior. An obvious statistical property of interest is the frequency with which various transitions in the state machine are taken. A self-propagating email virus will cause an increase in many of these frequencies. We may also be interested in statistical properties across a *subset* of instances, rather than all instances. The instances of interest can be specified on the basis of state variable values. For instance, we may be interested in the number of emails sent to any recipient by a particular user C on the network. We will do this by selecting instances that have `sender` equal to C in their `RCVD` state, and identifying the number of times the transition on the $deliver$ event was taken in these instances.

2.1 Statistics of Interest and their Representation

In the state machine in Figure 1, there are two significant transitions, taking place on the $send$ and $deliver$ events respectively. We therefore choose frequencies of these two transitions as statistical information of interest, and maintain the following statistics:

- frequency with which the $send$ transition is taken, across all clients
- frequency with which the $deliver$ transition is taken, across all clients
- for each client C, the frequency with which emails from C take the $send$ transition

– for each client C, the frequency with which emails *from* C take the $deliver$ transition

Each of these statistics were maintained at multiple (of the order of ten) time scales, ranging from about a second to about an hour.

We could maintain average frequency information, but since most phenomena related to email can be bursty, we choose to maintain *frequency distributions* rather than averages. In particular, we define a time window w over which we count the number of times a transition is taken. Let $n_{T_1}, ..., n_{T_k}$ denote the counts associated with a transition over k successive time periods $T_1, ..., T_k$ that are w units long. Then a histogram of the values $n_{T_1}, ..., n_{T_k}$ is used to represent the frequency distribution over a time window w, as observed during a training period of duration $w * k$ units.

Since we do not know in advance the range of the values n_{T-1}, it is more convenient to use a histogram with geometric bin ranges, i.e., the range of values corresponding to jth bin in the histogram is a times the range of the $(j-1)$th bin, for some factor a. In our experiments, a was set to $\sqrt{2}$. Thus, the histogram bins were $[0, 1), [1, 2), [2, 4), [4, 7), [7, 11), [11, 17), [17, 25)$ and so on.

As with other anomaly detection techniques, our approach consists of a training period, followed by a detection period. During the training period, a histogram H_t representing the frequency distribution observed during the training period is computed and stored. For detection, the histogram H_d computed during detection is compared with the histogram H_t. An anomaly is flagged if H_d is "more" than H_t. The notion of "more" can be defined in multiple ways, but we need some thing that can be computed efficiently, and moreover, represents a clear and significant change from H_d. For this reason, we compare the highest non-zero bin $HNZB_d$ in H_d with the highest non-zero bin $HNZB_t$ computed during training. The severity of the anomaly is defined to be $HNZB_d - HNZB_t$, provided $HNZB_d > HNZB_t$. Otherwise, no anomaly is flagged. The condition $HNZB_d > HNZB_t$ reflects our bias for detecting increased email traffic, as opposed to detecting a reduction in email traffic. Note that with this simple threshold criteria, there is no need to maintain entire histograms, but only the highest nonzero bins. More complex threshold criteria may take into account the entire histogram H_t to derive a threshold, so it is useful to compute and maintain histograms during training. During detection, however, the potential benefits of having the extra information will likely be more than offset by the additional storage needed to maintain histograms.

By choosing different values for the time window w, we can capture statistical information at different time scales. A small value of w will enable fast detection of intense attacks, as such attacks can be detected with a delay of the order of w. However, a slow but sustained attack may not be detected using a small time window. This is because there can be much more burstiness in email traffic at shorter time scales than larger time scales. Such burstiness means that the peak frequencies observed at shorter time scales will be much higher than average values, thus making it difficult to detect small increases in traffic. Since burstiness at higher time scales tends to be smaller, the difference between peak and average is smaller in those time scales, thus making it easier to detect modest increases in traffic. For this reason, we use several different time scales in our experiment, starting from 0.8 seconds and increasing to about 83 minutes, with each time scale being three to five times the previous one.

The above discussion separates the training phase from the detection phase. In a live system, user behaviors evolve over time, and this must be accommodated. The usual technique used in anomaly detection systems is to continuously train the system, while ensuring that (a) very old behaviors are "aged" out of the training profile, and (b) very recent behaviors do not significantly alter this profile. This technique can be incorporated into our approach as well, but we did not pursue this avenue as the change would have no direct effect on the results reported in this paper.

3 Experiment I

The primary goal of the first set of experiments was to study the effectiveness of our approach for detecting self-propagating email viruses. In particular, we wanted to study the false alarm rate and detection latency as the stealthiness of the virus is changed. This experiment us based on simple models of user behavior. (More complex and realistic user models are considered in Experiment II.)

One obvious way to study the effectiveness of the approach is to install it on a real mail server, such as the mail server in a university or a large company. Apart from issues of privacy that need to be addressed in such experiments, there is another serious impediment to such an approach: it is not practical to introduce viruses into such systems for the purpose of experimentation: it would seriously impact email service in the organization. Given the critical role that email has begun to assume in practically every large organization, such an approach is impractical.

Even if we were able to introduce such viruses in a real email system, existing email viruses are rather noisy: as soon as they are read, they send copies of themselves to all (or most) users in the address book. This causes a sharp spurt in email generation rate in the system, and would be immediately detected by our approach. To pose any challenge to our approach or to assess its capabilities, we would have to create new email viruses, which would be a significant task by itself. Therefore our experiment is based on simulation. Below, we describe the simulation environment, and proceed to present the results of the experiments.

An important aspect of these experiments is that the training, as well as detection took place in an unsupervised setting. No attempt was made to tune or refine the anomaly detector based on observed results. Such tuning or refinement could further improve the results.

3.1 Experimental Setup

For this experiment, we simulated an intranet with several hundred users. Three sizes of the intranet were considered: 400 users, 800 users and 1600 users. Our simulation could have been based on actual mail servers and mail clients that were driven by simulated users. However, the realism in the simulation is almost totally determined by the model used for user behavior, and is largely unaffected whether real email clients or mail servers were used[3]. On the other hand, leaving out real mail servers and clients in

[3] The only condition when the presence of real mail clients and servers can become important is when the system gets overloaded, due to propagation of email virus. In our experiments, the

a simulation has several important benefits. First, we do not need a large testbed consisting of hundreds of computers, real or virtual. Second, a light-weight simulation that avoids real mail servers and clients can complete in seconds instead of taking hours.

Our simulation used discrete time, where each cycle of simulation was chosen to correspond to roughly 0.2 seconds. This is a rather arbitrary number — our main concern in this context was to choose a small enough granularity that the results would be essentially the same as with a simulation based on continuous time.

Email users are modeled as Poisson processes, reading or sending emails at random during each simulation cycle. Specifically, in a single simulation cycle, the probability of a user sending email was set at 0.0006 and the probability of checking email was set at 0.0003. This means that users send out emails with a mean interval of about 5 minutes, and that they check emails with a mean interval of about 10 minutes. The recipients for each mail was determined at random, and the number of recipients was chosen using a positive normal distribution with a mean of 1 and standard deviation of 2. Whereas sending of mails was assumed to take place one at a time, email reading was modeled as a batch process — each attempt to read email reads most of the emails queued for the user. Moreover, for each message, the user randomly chooses to reply to the sender, reply to all recipients, or not reply at all. We have used identical models for all users in this experiment, while the experiment described in Section 4 uses a non-uniform model where different user behaviors are different.

In this experiment, we wanted to model not only the viruses prevalent today, all of which propagate very rapidly, but also stealthy viruses. For stealth, viruses may employ a combination of the following techniques:

- *low propagation factor,* i.e., when the virus is read, it does not cause generation of emails to a large number of users, such as the set of names in the address book of the reader. A high propagation factor makes the virus much more noticeable.
- *long incubation period,* the delay between when the virus is read and the time it causes propagation of email is large. The long delay makes it difficult to associate the propagation with the virus.
- *polymorphism,* the virus modifies itself, so that the emails generated do not look like the virus that was read. For additional stealth, the virus can propagate non-virus carrying emails as well as those carrying the virus.
- *matching user behavior,* i.e., the virus avoids sending out emails with a large recipient list, instead partitioning such messages into multiple ones with recipient lists of the size observed on normal messages.
- *randomization,* i.e., all of the above techniques are randomized — for instance, the incubation period is a random number over a range. Similarly, the propagation factor is a random number.

Of these techniques, polymorphism does not affect our approach, as it is not based on email content. Among the rest, propagation factor and incubation period were found to have the maximum impact on detection effectiveness, while randomization had mod-

---

virus was always detected well before there was any significant increase in email traffic, and hence the absence of actual email servers and clients is unlikely to have affected the results we obtained.

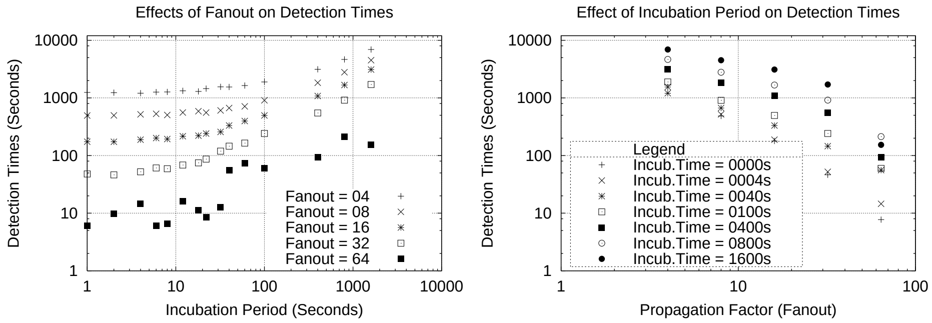

Fig. 2. Detection time as a function of incubation period and propagation factor.

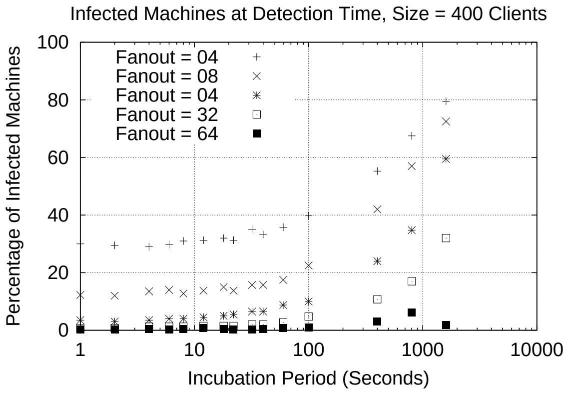

Fig. 3. Percentage of Infected Hosts

est effect. Matching of user behavior seemed to have no effect. Thus, our results discussion considers only two of the above factors: propagation factor and incubation period.

3.2 Metrics used for detection

The first and most obvious metric is the detection time: the time between the introduction of the virus and the time of detection. Figure 2 shows how the detection time changes as the propagation factor (also known as fanout) and incubation period are changed. Note that longer incubation periods and lower propagation factors delay detection. The detection delay is somewhat mitigated by the fact that the virus itself propagates more slowly in these cases. We therefore look at other metrics that factor out the speed at which a virus spreads. Some of these metrics are:

– percentage of clients that are infected at the time of detection
– percentage of email traffic due to viruses at the time of detection

The first of these metrics is related to the costs for cleaning up after the virus infection. The other metric relates to the load on the email server, and the degree to which its function is degraded by the virus.

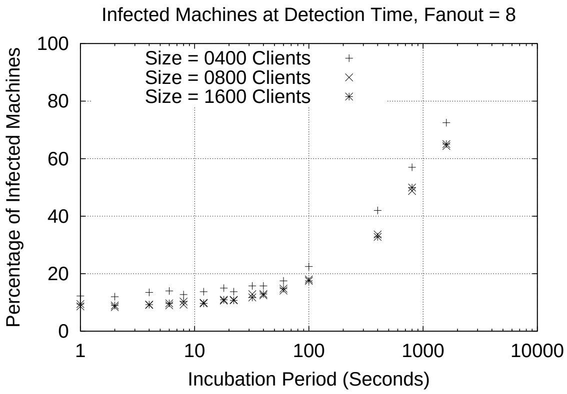

Fig. 4. Infected Hosts

Figure 3 shows the percentage of infected hosts at the time of detection of the attack. The results are for an intranet consisting of 400 clients. This figure shows that for noisy viruses, detection occurs early, but very stealthy viruses, especially those that use a combination of large incubation periods and low propagation factors, can potentially infect most of the network before being detected. Figure 4 shows that for a given value of propagation factor (fixed at 8 for this graph), and incubation period, the fraction of infected hosts is lower when the number of clients in the intranet is higher.

Figure 5 shows the fraction of email traffic that is due to the viruses as of the time of detection. Specifically, we calculated the fraction of email traffic due to viruses in the few seconds (2 seconds) preceding the detection. Note that the virus traffic is in the 40% to 70% range, which means that the email server is only slightly overloaded. Due to burstiness of emails, servers are typically designed to handle a few to several times the average rate at which emails are generated in the system. For this reason, a 40% to 70% increase in email traffic is not very significant.

3.3 False alarms

False alarm rates were computed using two different criteria:

- *Criteria 1*: Count even a single alarm as a false alarm: Using this criteria, there were a total of 3 false alarms across 8 runs, or a rate of about 0.38 false alarms per hour.
- *Criteria 2*: Apply a threshold criteria, and count a false alarm when the threshold is exceeded. This threshold is established through experimentation. We found that by registering an alarm when more than 3 alarms are reported over a period of two seconds, zero false alarm rate could be achieved in our simulation.

We note that in the detection results reported earlier, Criteria 2 was used. Thus, those detection results were obtained with zero false alarm rate.

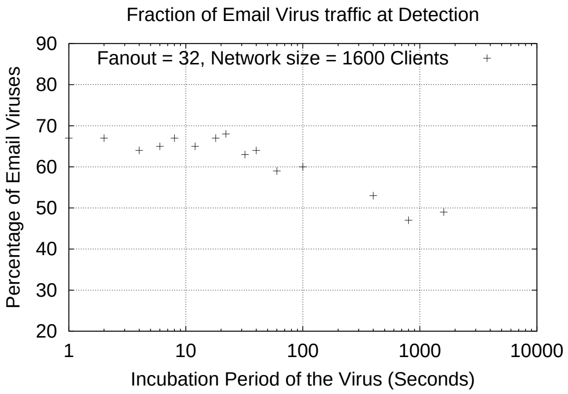

Fig. 5. Percentage of Virus Carrying Emails

Runtime Performance and Memory Usage

The whole implementation was done in Java. With 400 clients, about 800 frequency distributions were maintained, each over 8 time scales. Due to these structures the total memory use of the Java program was 30MB. When run on a Intel Pentium III system operating at 1GHz, it was able to simulate about 500 cycles per second, i.e., simulate 100 seconds in one second of operation. In addition to the simulation, the anomaly detector was processing about 100 messages per second. This performance was adequate to provide fast simulation. If used in a live environment, these performance results show that the anomaly detector will consume 1% of CPU on a similar system.

4 Experiment II

This experiment was conducted as part of the DARPA SWWIM program. The SWWIM Autonomic Response Architecture (SARA) experiment was conducted by a collaborative team of organizations, each responsible for a key function. This experiment differed from the previous experiment in several aspects. First, the user models were asymmetric, i.e., the behavior models for different users were different. Second, the experiment was conducted with real email servers (sendmail) and clients. Third, the simulation as well as the viruses were designed by a third party that had no vested interest in how the detectors from different organizations performed.

The overall goal of the SARA experiment was to evaluate the value of orchestrated response to attacks. The system consisted of several virus detection components, response components in the form of mail server and client enhancements to purge suspected messages, and an orchestrator. The orchestrator took its input from the detection components, evaluated the system state based on these inputs, selected a response action, and communicated these actions to the response components. Several detection components were built, including (a) simple behavior based detectors that looked for more than a certain number emails within a certain time period or within a certain time period after an attachment was opened, (b) more complex behavior based detectors that were tuned to detect the tree-like flow of emails produced by email viruses, and (c) our anomaly detector.

Early on in the experimental design, it was decided that the above detectors would be used in different stages of virus spread: the behavior based techniques will be used for early detection, at which point the system would attempt a carefully orchestrated sequence of responses. But these detectors can be fooled by stealthy viruses, at which point, the results from the anomaly detector would be used to identify the spread of the virus. Note that the anomaly detector cannot provide precise identification of offending email messages — the only thing that can be said is that a predominant number of email messages causing an alarm are bound to be viruses. Due to the absence of precise identification of virus-carrying emails, and given the time constraints associated with the conduct of this experiment, it was decided that the orchestrator would simply shutdown the system if the only information it had was from the anomaly detector. Clearly, this is a response of last resort, and not to be attempted unless every thing else failed. In particular, the orchestrator should be allowed to try intelligent responses based on inputs from other detectors; and only when all of this failed, it should consider shutting down the system. In order to make sure that these responses were given adequate time to work, it was decided that the anomaly detector would artificially delay detection until such time it became clear that the virus was established in spite of an orchestrated response.

4.1 Experimental Setup

The experiment was carried out using a "full scale" simulation of an email system for a single subnet of 400 clients. This included an email server (modified version of sendmail) and 400 email clients. The detection, response, and orchestration components communicated and worked in conjunction with the email server and clients.

Similar to Experiment I, the actions of users were emulated by 400 "bots." However, these bots were significantly more complex than user models used in Experiment I. In particular, user behavior was simulated by 400 bots that were implemented as processes that run concurrently. User behavior was modeled using a three-state Markov model, with the states corresponding to the user reading email, composing email and being idle. The bots will make transitions at random among these states, with a specified probability for each of the six possible transitions. In this manner this model avoids the pitfalls associated with a Poisson model used in Experiment I.

A second important improvement in the user model is that it is asymmetric, and it captures the concept of *address books.* When a user composes email, the set of recipients is assumed to come from his or her address book. The address book size is unlimited, i.e., it can be as large as the user population. These factors mean that it is much more common for emails with a large number of recipients to be generated in this experiment.

Several different types of viruses were used in the experiment. These virus types are shown in Figure 6. Higher numbered viruses were intended to be progressively more stealthy.

Virus type	Description
1	Static
2a	Randomized Addresses - (taken from sent items)
2b	Randomized Addresses - (taken from received items)
3a	Randomized - (random number of recipients)
3b	Delayed Randomized (random no. of recipients and time delay)
4a	Polymorphic - (virus attachments all end in .vbs)
4a.v1	Polymorphic - (virus attachments have variable extensions)
4b	Persistent Polymorphic - (virus attachments all end in .vbs, lives forever)
4b.v1	Persistent Polymorphic - (fast propagating version)
4b.v2	Persistent Polymorphic - (slow propagating version)
4b.v3	Persistent Polymorphic - (medium propagating version)
4b.v4	Persistent Polymorphic - (viruses have variable extensions, lives forever)

Fig. 6. Properties of Viruses Used

4.2 Detection Effectiveness

Hundreds of simulation runs were carried out with the above types of viruses. Due to the fact that the anomaly detector was tuned explicitly for delayed detection of viruses, no alarms were generated in those runs where the orchestrator was able to contain the virus. There were seven runs in which the orchestrator was unable to contain the virus. It is significant that in every one of these cases, the anomaly detector was able to detect the virus, as shown in Figure 7. In most cases, the detection took place 2 or 3 minutes after the detection of virus.

Virus type	Time of (post-virus release) detection	Percentage of traffic consumed by virus
2b	3.7 min	$<$ 5 percent
4a	36.4 min	$<$ 5 percent
4b	3.0 min	$<$ 5 percent
4b	2.2 min	$<$ 5 percent
4b	3.3 min	$<$ 5 percent
4b	3.1 min	$<$ 5 percent
4b.v2	22.7 min	$<$ 5 percent

Fig. 7. Virus Detection

In some cases, the detection was rather slow. For virus 4b.v2, the delay was due to the fact that it had a very long incubation period, so it was not propagating fast until around 20 minutes after its introduction. Thus, detection took only two minutes after the virus became active. In the case of virus 4a, the orchestrator was initially able to contain the virus, and hence no alarms were reported by the anomaly detector. However, after about 30 minutes of containment, the orchestrator lost control of the virus, which subsequently took over the system. The detection occurred a few minutes after the point when the virus got away.

4.3 False alarm analysis:

As for Experiment I, false alarm rates were measured in two ways:

- *Criteria 1*: Count even a single alarm as a false alarm: Using this criteria, there were a total of 18 false alarms across 6 runs, or a rate of about 0.3 false alarms per hour. (Compare this to the 0.38 false alarms per hour obtained using this same criteria in Experiment I.)
- *Criteria 2*: Set a threshold via experimentation. In this case, the threshold was set so that not only do we mask false alerts, but also true alarms that are not sufficiently severe to warrant a system shutdown. (Recall that the only response used in the experiment was to shutdown the mail server when the anomaly detector produced an alarm.) For this reason, the threshold was much higher than in Experiment I. Specifically, we identified a threshold of 50 or more alarms in a period of 256 seconds. Using this criteria, no false alarms were observed. (In fact, the maximum number of alarms produced within a period of 256 seconds in any of these six runs was 14, which is well below the 50 threshold.)

We note that in the detection results reported in Figure 7, Criteria 2 was used. Thus, those detection results were obtained with zero false alarm rate.

4.4 Runtime Performance and Memory Usage

The anomaly detector performance and memory usage in this experiment was similar to that reported for 400 clients in experiment I.

5 Related Work

Self-propagating malicious programs have been analyzed ever since they came into existence starting with the Morris worm [2]. Along with the growth of the Internet, the threat of worms spreading into computer networks has also increased. To understand and predict the propagation of such worms has become an increasingly important research topic. Propagation analysis and detection has also been carried out for more recent Code Red [12] and Melissa [1] viruses, where the email is used as the vehicle of propagation for these malicious executables.

Incidents of virus propagation through the cyber realm have been viewed and modeled using epidemiological modeling, mapping the Internet to mathematical models of ecological systems [15]. Models have been developed to accurately predict the propagation of worms and viruses through the networks. One such example is a variation of Kermack-Mckendrick model, used to predict the propagation of the Code Red virus through the Internet [39]. At IBM, Kephart and White have developed systems for detection using these models [7],[8] [9]. In addition to borrowing ideas from mathematical epidemiology, the model has been extended by incorporating network topological effects, using power-law relationships [20] which try to give some structure to the apparent randomness of the Internet. [10] studies the propagation of viruses when a subset of the hosts are immune to the virus. [18] studies the problem of network availability in the face of viral attacks. The focus of all these efforts were to study the propagation of viruses, whereas the focus of this paper is the development of an effective detection technique.

Anomaly detection techniques have long been used for intrusion detection [13, 27, 25, 32–34, 16, 36]. The approach developed in this paper is closely related to [36]. In both approaches, a protocol state machine specification forms the basis for detection. This state machine is used to transform events (such as network packets, or sending or delivery of emails) into frequency distributions that characterize normal behavior. The training and detection phases are robust, and can operate without any supervision. These factors contrast with most other anomaly detection approaches, especially at the network level, where considerable knowledge and ingenuity was needed to identify the set of "features" to be included in normal behavior characterization. Moreover, many of these techniques required expert supervision to make sure that the normal behavior characterization learned by the technique was indeed appropriate.

The Malicious Email Tracking (MET) system [17] was developed to track the flow of malicious emails such as self-replicating viruses through a network. It was designed as a system to track flow of malicious email traffic on wide area network without having to sample most of the emails exchanged in the network. However, its techniques for detecting malicious emails, such as the use of MD5 sums for identification of the propagation of the same virus, can be defeated by polymorphic viruses such as those considered in this paper.

While MET is focused specifically on emails, the earlier Graph-based intrusion detection system (GrIDS) [31] work was focused on the more general problem of large-scale automated attacks that propagate over the network. GrIDS is based on assembling the activities on different network nodes into *activity graphs* that show the spread of attacks through a network. It can also support policy-based detection of attacks by detecting policy violations in the activity graph.

[6] uses a data mining approach to detect malicious executables embedded within emails. Short sequences of machine instructions are the features used in this approach. A Naive Bayes classifier, trained on a set of malicious and a set of benign executables, was used to detect whether an attachment contained malicious code. This approach assumes that there are similarities among the binary code of malicious executables. While this is shown to be true for viruses known today, it is easy enough to write stealthy viruses that can escape detection by this technique.

The Email Mining Toolkit (EMT) [35] work complements MET in that it uses data mining to synthesize the behavior profiles of users that is used by MET to detect malicious email. EMT models "normal behavior" of each email user in terms of several characteristics such as the identities of the other users they communicate with, and the frequencies with which they communicate with these users. It can detect not only viruses, but also changes in communication patterns that may result due to misuse or other malicious user behavior. However, for the purpose of virus detection, this technique is likely to have higher latency than the technique proposed in this paper. This is because the sending of a single message, or even a few virus messages, cannot be considered a significant departure from normal communication pattern without increasing the false alarm rate.

6 Conclusions and Future Work

In this paper, we presented a new technique for detecting self-propagating email viruses using statistical anomaly detection. Our results suggest that the kinds of viruses prevalent today can be detected before a significant fraction of the network is infected. Our approach degrades gracefully when facing more stealthy viruses that use a combination of low propagation factor, high incubation period and randomization. We note that an email virus writer has to be careful in designing a stealthy virus: if it uses too low a propagation factor, then it may "die" in the presence of hosts that are immune to the virus (e.g., Microsoft Outlook viruses sent to Netscape or Lotus Notes users). A high incubation period also delays the spread of the virus, which provides more opportunities for a vigilant user or system administrator to notice the virus. Thus it is likely that very stealthy viruses are not very stable.

When we began this work, we assumed that an anomaly detection technique such as ours will have a significant latency in detection, by which time most of the network may be infected. While this assumption turned out to be true for the most stealthy of the viruses used in our experiments, our results suggest that for a majority of the viruses, it is potentially feasible to detect attacks when only a minority of the network is infected. Note that with early detection, the costs associated with cleaning up such viruses can be reduced.

While the results presented in this paper are promising, their main weakness is that they are all based on simulation. Real systems often display behaviors that are more complex and variable than those exhibited in simulations. This factor can artificially inflate the effectiveness of anomaly detection systems during simulations. In order to really assess the effectiveness of the approach, it is necessary to evaluate it using realistic email traffic. Our ongoing work develops techniques where email traffic is no longer simulated, but is taken from mail server logs. The virus models will continue to be simulated. The traffic presented to the anomaly detector is obtained by superimposing the background traffic from the logs with simulated virus email traffic.

A second difficulty in extrapolating the simulation results is that on real systems, email traffic crosses organization boundaries frequently. In particular, a virus may propagate from one user to any other user on the Internet, and not just on the intranet of the user's organization. At the same time, it is not realistic to assume that our anomaly detector can be deployed Internet-wide. Thus, a question arises as to how well an Internet-wide virus propagation can be detected by an anomaly detector observing the behavior of email on an intranet. This is another question that needs to be addressed in future research.

References

[1] CERT/CC Co-ordination Center Advisories, Carnegie Mellon, 1988-1998, http://www.cert.org/advisories/index.html.

[2] Eugene H. Spafford, The Internet worm program: an analysis, Tech. Report CSD-TR-823, Department of Computer Science, Purdue University, 1988.

[3] Terran Lane, Carla E. Brodley, Temporal Sequence Learning and Data Reduction for Anomaly Detection, ACM Transactions on Information and System Security, 1998.

[4] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, T. Garvey, A real-time intrusion detection expert system (IDES) - final technical report.

Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

[5] T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, D. Wobler, A Network Security Monitor , Proceedings IEEE Symposium on Research in Computer Security and Privacy, 1990.

[6] Matthew G. Schultz, Eleazar Eskin, and Salvatore J. Stolfo. "Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables." Proceedings of USENIX Annual Technical Conference, 2001.

[7] J.O. Kephart, S.R. White, Directed-graph Epidemiological Models of Computer Viruses, IBM T.J. Watson Research Center, IEEE Computer Society Symposium on Research in Security and Privacy, pp. 343-359, 1991.

[8] J.O. Kephart, David M. Chess, S.R. White, Computers and Epidemiology, IBM T.J. Watson Research Center, IEEE Spectrum, May 1993.

[9] J.O. Kephart, G.B. Sorkia, M. Swimmer, S.R. White, Blueprint for a Computer Immune System. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1997.

[10] Chenxi Wang, John C. Knight, Matthew C. Elder, On Computer Viral Infection and the Effect of Immunization, Department of Computer Science, University of Virginia, ACSAC 2000.

[11] Klaus Julisch, Mining Alarm Clusters to Improve Alarm Handling Efficiency, IBM Research, Zurich Research Laboratory, ACSAC 2001.

[12] Stuart Staniford, Analysis of spread of July infestation of the Code Red worm, UC Davis, http://www.silicondefense.com/cr/july.html.

[13] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes, Next-generation Intrusion Detection Expert System (NIDES): A Summary, SRI-CSL-95-07, SRI International, 1995.

[14] S. Staniford, V. Paxson, N. Weaver, How to Own the Internet in Your Spare Time, Usenix Security Symposium, 2002.

[15] Jane Jorgensen, P. Rossignol, M. Takikawa, D. Upper, Cyber Ecology: Looking to Ecology for Insights into Information Assurance, DISCEX 2001. Proceedings , Volume 2 , 2001.

[16] Carol Taylor, Jim Alves-Foss, NATE, Network Analysis of Anomalous Traffic Events, A Low-cost Approach, New Security Paradigms Workshop, 2001.

[17] Manasi Bhattacharyya, Shlomo Hershkop, Eleazar Eskin, and Salvatore J. Stolfo, MET: An Experimental System for Malicious Email Tracking, Workshop on New Security Paradigms, 2002 (NSPW-2002).

[18] Meng-Jang Lin, Aleta M. Ricciardi, Keith Marzullo, A New Model for Availability in the Face of Self-Propagating Attacks, Workshop on New Security Paradigms, 1998.

[19] Wenke Lee, Salvatore J. Stolfo, Kui W. Mok, A Data Mining Framework for Building Intrusion Detection Models, IEEE Symposium on Security and Privacy, 1999.

[20] M. Faloutsos, P. Faloutsos, C. Faloutsos, On Power-Law Relationships of the Internet, ACM SIGCOMM, 1999.

[21] M.G. Schultz, E.Eskin, E. Zadok, Data Mining Methods for Detection of New Malicious Executables, IEEE Symposium on Security and Privacy, May 2001.

[22] Ian Whalley, Bill Arnold, David Chess, John Morar, Alla Segal, Morton Swimmer, An Environment for Controlled Worm Replication and Analysis, IBM TJ Watson Research Center, Sept 2000.

[23] L. Heberlein et al, A Network Security Monitor, Symposium on Research Security and Privacy, 1990.

[24] J. Hochberg et al, NADIR: An Automated System for Detecting Network Intrusion and Misuse, Computers and Security 12(3), May 1993.

[25] W. Lee and S. Stolfo, Data Mining Approaches for Intrusion Detection, USENIX Security Symposium, 1998.

[26] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time, USENIX Security Symposium, 1998.

[27] P. Porras and P. Neumann, EMERALD: Event Monitoring Enabled Responses to Anomalous Live Disturbances, National Information Systems Security Conference, 1997.

[28] Inc. Network Flight Recorder. Network flight recorder, http://www.nfr.com, 1997.

[29] G. Vigna and R. Kemmerer, NetSTAT: A Network-based Intrusion Detection Approach, Computer Security Applications Conference, 1998.

[30] G. Vigna, S.T. Eckmann, and R.A. Kemmerer, The STAT Tool Suite, in Proceedings of DISCEX 2000, IEEE Press, 2000.

[31] Staniford-Chen, S. et al, GrIDS: A Graph-Based Intrusion Detection System for Large Networks. Proceedings of the 19th National Information Systems Security Conference, Baltimore, 1996.

[32] S. Forrest, S. Hofmeyr and A. Somayaji, Computer Immunology, Comm. of ACM 40(10), 1997.

[33] A. Ghosh, A. Schwartzbard and M. Schatz, Learning Program Behavior Profiles for Intrusion Detection, 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 1999.

[34] R. Sekar, M. Bendre, P. Bollineni and D. Dhurjati, A Fast Automaton-Based Approach for Learning Program Behaviors, IEEE Symposium on Security and Privacy, 2001.

[35] Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern, Chia-Wei Hu, Behavior Profiling of Email, submitted to 1st NSF/NIJ Symposium on Intelligence and Security Informatics(ISI 2003).

[36] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, S. Zhou, Specification-based anomaly detection: a new approach for detecting network intrusions, ACM Computer and Communication Security Conference, 2002.

[37] R. Sekar, Y. Guang, T. Shanbhag and S. Verma, A High-Performance Network Intrusion Detection System, ACM Computer and Communication Security Conference, 1999.

[38] R. Sekar and P. Uppuluri, Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications, USENIX Security Symposium, 1999.

[39] C.C. Zou, W. Gong, Don Towsley, Code Red Worm Propagation Modeling and Analysis, ACM Computer and Communication Security Conference, 2002.