

An Email Worm Vaccine Architecture

Stelios Sidiroglou, John Ioannidis, Angelos D. Keromytis, and Salvatore J. Stolfo

Department of Computer Science, Columbia University
{stelios, ji, angelos, sal}@cs.columbia.edu

Abstract. We present an architecture for detecting “zero-day” worms and viruses in incoming email. Our main idea is to intercept every incoming message, pre-scan it for potentially dangerous attachments, and only deliver messages that are deemed safe. Unlike traditional scanning techniques that rely on some form of pattern matching (signatures), we use behavior-based anomaly detection. Under our approach, we “open” all suspicious attachments inside an instrumented virtual machine looking for dangerous actions, such as writing to the Windows registry, and flag suspicious messages. The attachment processing can be offloaded to a cluster of ancillary machines (as many as are needed to keep up with a site’s email load), thus not imposing any computational load on the mail server. Messages flagged are put in a “quarantine” area for further, more labor-intensive processing. Our implementation shows that we can use a large number of malware-checking VMs operating in parallel to cope with high loads. Finally, we show that we are able to detect the actions of all malicious software we tested, while keeping the false positive rate to under 5%.

1 Introduction

Recent incidents have demonstrated the ability of email-based worms and viruses (“malware”) to infect large numbers of hosts very rapidly [1, 2]. Email malware propagates as executable attachments that users are tricked into opening, thus causing the malignant code to run and propagate, usually by sending copies of itself to all the entries in the user’s address file. While email attachments are not the only vector by which malware propagates, they pose a substantial threat that merits special treatment, especially since attachments have the advantage (from the defender’s perspective) that they can be caught before they hit the user’s machine. There are numerous approaches to defending against malicious software, the usual example being the various antivirus packages.

Virus scanners are predominately signature-based, identifying security threats by scanning files for certain byte sequences that match already known patterns of malicious code. This translates to a constant need for maintaining an up-to-date signature database. The problem is further exacerbated by the lag in the cycle of detecting a new attack and the deployment of the corresponding signature, especially when humans are involved in the process. Many modern email-borne viruses do not rely on software bugs; rather, they rely on humans to click on the attachments, thus activating them.

The need for frequent updates and the inherent delay between the creation of malicious software, and the detection and deployment of signatures or patches relegate signature-based techniques to a secondary role in the active security of systems.

Behavior-based mechanisms characterize software based on the perceived effects that the program has on the examined system, instead of relying on distinct signatures of that software. The obvious benefit of this approach is that it can detect new attacks (no prior knowledge or signatures required) as long as there is some differentiation between the behavior of a malicious and normal piece of software. The majority of these behavior-based systems rely on anomaly detection algorithms for their classification and thus detection of malignant code. Anomaly-detection algorithms work by constructing models of normal behavior and subsequently checking observed behavior against these models for any statistically significant variations that may hint at malicious behavior. The success of an anomaly detection algorithm depends on the choice of an accurate behavior model. Current host-based IDS systems employ anomaly detection algorithms that are based on network activity, system call, and file system monitoring. The reason behind the absence of reliable host-based IDS that are based on the fore-mentioned models has primarily to do with the overbearing computational overhead associated with extracting behavior models from irregular and high-volume events. In particular, analyzing all system calls in a system imposes a considerable overhead due to the sheer volume of events; correlating this with the highly irregular nature of system calls in general imposes a considerable computational overhead with a high false positive rate as a further disadvantage.

Our approach combines the ability of a host-based IDS to detect previously unseen malware with the concept of a mail-server based filtering solution. To wit, we scan each incoming mail message at the mail server for potentially dangerous attachments. Every such attachment is sent to one of a set of protected environments running various mail readers (MUAs — Mail User Agents) along with a host-based IDS. In our particular instance, the IDS looks for registry accesses which the MUA is not likely to perform. Using the Windows registry allows for a more accurate correlation of events given that it is an integral component of the Windows operating system and helps reduce the false positive rate associated with detecting malicious behavior. The protected environment opens each executable attachment, runs it, and if the IDS detects suspicious behavior, it notifies the mailserver to discard the corresponding email message. The entire environment is running under VMware so that no clean-up needs to be performed; the VM is discarded and a new one is spawned for each new check.

The advantage of such an approach is that adding compute power (faster and/or more machines) to the checking components allows a site to customize the resources needed for defense to its needs. Different environments can be set up running different MUAs, selected based on the local user population. Such an approach also does not preclude using traditional techniques such as pattern-matching to catch known viruses.

Our implementation shows that we can use a large number of malware-checking VMs operating in parallel to cope with high loads. The average time for downloading the message, detecting the attack and updating the MTA message queue was 28 seconds. Finally, we show that we are able to detect the actions of all malicious software we tested, while keeping the false positive rate to under 5%. Combining additional detectors will enable the use of data correlation algorithms that can be used, in turn, to reduce the false positive rate.

2 System Architecture

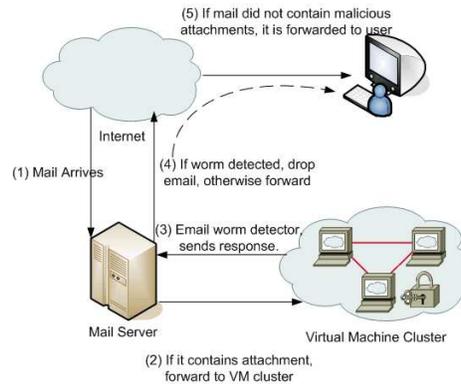


Fig. 1. System Architecture

Our architecture makes use of several components that have been developed for other purposes. Its novelty lies in the combination of all the components in detecting and deterring zero-day email worms from propagating using an entirely automated process. As illustrated in Figure 1, our system consists of three main components:

- A virtual machine cluster, which houses protected environments that run instances of different Mail User Agents (MUAs) and operating systems.
- A host-based IDS that is responsible for detecting anomalous behavior.
- An email-worm-vaccine aware Mail Transport Agent (MTA) that classifies and manages potentially malicious email messages.

In the following sections we discuss the individual components in detail.

2.1 Virtual Machine

Host-based Intrusion Detection Systems (IDS) must, by definition, run the potentially malicious application on the host machine. Allowing the attack to run locally renders that particular machine useless for further use. For this reason, the malicious software is tested on an isolated, controlled environment that will provide the level of protection required. An ideal candidate is a virtual machine environment that can be effectively flushed after each use without further impact to the underlying system. Specifically, we use VMware images that contain the base system that we use across the VM-cluster, which has the advantage of providing an identical test case for use with the host-based IDS. An additional benefit of using a centralized VM-based architecture is that we avoid the need to deploy IDS and mail filtering software on large numbers of desktops.

2.2 Host-based Intrusion Detection

In order to be able to detect zero-day email worms, we need a non signature-based approach. For this purpose, we employ a behavior based mechanism as the anomaly detection component of our architecture.

Behavior-based mechanisms extract and characterize software based on the perceived effects that the program has on the examined system in lieu of relying on distinct signatures of that software. The obvious benefit of this approach is that it can detect new attacks (no prior knowledge or signatures required) as long as there is some differentiation between the behavior of a malicious and the behavior of a normal piece of software. The majority of these behavior-based systems rely on anomaly detection algorithms for their classification and thus detection of malignant code. Anomaly detection algorithms work by constructing models of normal behavior and subsequently checking observed behavior against these models for any variations that may hint at malicious behavior. As mentioned earlier, the success of an anomaly- detection algorithm is contingent upon the choice of behavior model. Current host-based IDS systems employ anomaly detection algorithms that are based on network activity, system call and file system monitoring. The reason behind the absence of reliable host-based IDS that are based on the aforementioned models has primarily to do with the high computational overhead associated with extracting behavior models from irregular and high-volume events. In particular, analyzing all system calls in a system imposes an considerable overhead due to the sheer volume of events; correlating this with the highly irregular nature of system calls in general imposes a considerable computational overhead with the a high false positive rate as appendage.

2.3 MTA

Another critical component of our architecture is an email-worm-vaccine aware Mail Transfer Agent (MTA). The purpose of this augmented MTA's components are:

- Classification and filtering of potentially malicious email
- Communication with the host-based IDS cluster
- Maintenance of message queue

The MTA, as a first line of defense, will be responsible for imposing message classification and filtering. A tightly-coupled learning component will facilitate the decision process by receiving feedback from the host-based IDS. The filtering component of the MTA will conceptually reside in front of the classification component. Filtering will be the primary means by which to avoid denial-of-service attacks on the underlying system. For example, in the case of a mass email-worm outbreak, once the IDS component identifies a message as containing a malicious payload all subsequent email containing identical payloads will be sent directly to the quarantine component, bypassing the rest of the system. This case becomes much more difficult to solve for polymorphic and metamorphic email-worms; Spinellis [3] shows that it is an NP-hard problem. The only viable plan of action in the presence of a high-volume polymorphic outbreak would be to filter all incoming email that fit the high-level characteristics (having an attachment

or originating from a particular source) by either pushing them directly to the quarantine or replying with a 451 (“transient error, try again later”) message.

Classification of messages would be performed on the basis of a set of heuristics such as the presence of attachments or embedded URLs. Once a message has been classified as suspicious, it is sent to the host-based IDS VM cluster. At that point, the messages are placed in temporary queues waiting for a decision from the IDS.

2.4 Mail User Agent

The final component of the system architecture is the Mail User Agent (MUA). The primary purpose of the MUA is the retrieval and execution of potentially malicious email. The MUA, in turn, simulates the behavior of a naïve user by opening all email attachments and clicking on all available URLs. The reason that we use an MUA instead of simply downloading the email directly is so that we can expose any vulnerabilities that are directly related to using the particular MUA.

3 Implementation

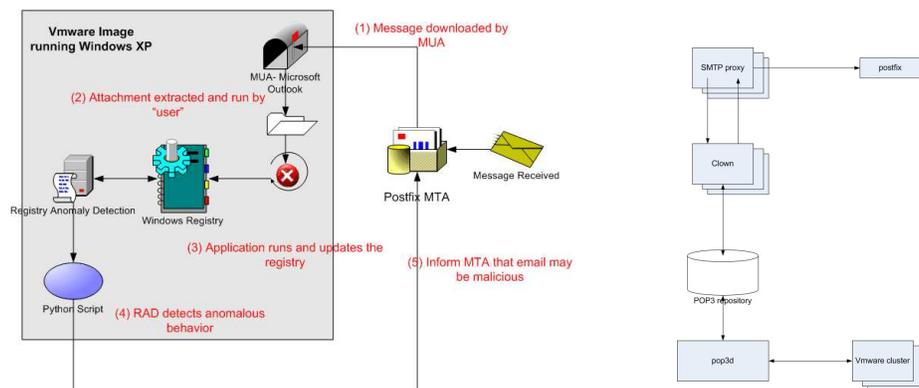


Fig. 2. System Implementation (left), with MTA details (right)

Our prototype implementation, shown in Figure 2(left) consists of four components: RAD [4], VMware [5], Postfix [6] and Microsoft Outlook [7]. These components interact to provide a secure environment, detect anomalous behavior, manage email queues and simulate naive user behavior respectively. In Section 4 we evaluate the performance of our approach. Here, we introduce the components and discuss the implementation.

3.1 RAD

In order to detect anomalous behavior, namely email worms, we employ a RAD (Registry Anomaly Detection) [4], which monitors in real-time accesses to the Windows registry and detects malicious behavior.

The Windows Registry is an integral component of the Windows operating system, frequently used by a majority of programs. These characteristics elevate the Registry to prime candidate position as source of audit data. RAD attaches a sensor on the Registry and applies the acquired information to an anomaly detector that can correlate activity that corresponds to malicious software.

The main advantage of using RAD is its ability to accurately detect anomalous behavior with a low computational overhead. The low overhead makes it a viable solution for real-time detection of malicious software.

RAD constructs a data model from five features extracted directly from the registry sensor. These features are: the name of the process accessing the registry, the type of query sent to the registry, the key that is being accessed, the response from the registry, and the value of the key that is being accessed.

Using the features thus monitored from the registry accesses, RAD builds a model from normal (non-attack) data. This model is then used to classify registry accesses as either normal or malicious.

3.2 VMware

VMware allows multiple virtual machines, each running its own operating system, to co-exist on a single real machine. Potentially dangerous applications can thus be isolated from each other by running them in separate virtual machines. We prepare a single VMware image that contains an already trained model for our host-based IDS and the applications that we are testing, namely, standard Microsoft products (Office, Outlook, Outlook express, Messenger) and various other popular applications.

The image is used for a single detection session; testing a single email attachment at a time. For this purpose we set the VMware disk mode to “non-persistent, so that any changes made to “disk” are lost when the virtual machine is terminated. Having the disk in nonpersistent mode allows for one additional advantage, the use of the repeatable resume feature. Repeatable resume allows for a virtual machine to quickly start from a resumed state bypassing the need to reboot the operating system any time a new virtual machine environment is needed.

3.3 MTA

We based our implementation on the `smtp.proxy` open-source package as a front-end for any MTA. Figure 2(right) shows the components of this implementation. `smtp.proxy` is a relatively simple piece of code that listens on the SMTP port (port 25), waiting for incoming SMTP connections. When a connection arrives, the proxy contacts the real MTA (in our case, Postfix [6]) and goes through the initial HELO/MAIL/RCPT phase

with both sides. Thus, our proxy does not have to know any special site-specific restrictions on acceptable domains, anti-spam measures, and so on, that the Postfix administrator may have set up. Configuration details such as preventing open-relays or maximizing concurrency are beyond the scope of this paper — ours is a proof-of-concept, not a highly-optimized implementation. When the remote MTA sends the `DATA` command, followed by the body of the email message, the proxy saves it in a uniquely-named temporary file, and invokes a script which we wrote, *clown*, after it has received the entire message, but before it responds to the `DATA` command of the remote MTA.

A copy of *clown* is forked for every message received; it therefore keeps a tally of how many copies of itself are currently running, waiting for the cleanup VMs to return. If a limit, chosen so that the queue of unprocessed messages does not grow steadily, is exceeded, *clown* returns a 451 (“transient error, try again later”) message, which causes *smtp.proxy* to pass that on to the remote MTA so that the mail message can be processed later. The local copy is then removed.

Once *clown* receives control, it runs the file with the contents of the email message through a MIME normalizer (a separate big problem in itself, and outside the scope of this paper); it then passes a copy of the message on to one of the cleanup virtual machines and waits for the VM to finish processing. The copy passed to the VM includes an extra header with the IP address and port to contact (e.g., `X-Clown: 128.59.16.20:12588`). The VM will respond with an indication as to whether the message is acceptable or not. If the message is deemed safe, *clown* will simply return with a 0 exit code, at which point *smtp.proxy* will pass the file on to the real MTA for eventual delivery. Otherwise, a 554 (“permanent error”) response will be given to the proxy, which will pass it on to the remote MTA. The copy of the message is discarded, *clown* exits, and another queued message is processed.

We had a choice between a pull-model and a push-model for passing the messages on to the VM cluster. We opted for the pull-model, as it made implementation easier. To wit, *clown* deposits every message in a POP3 repository, which, for the particular POP3 server we use, happens to be the Unix mail file format. As each VM becomes available, it pulls the topmost (oldest) message from the POP3 server, processes it, and then connects to the TCP port specified in the `X-Clown:` header.

To ward against VM cluster failures or excessive load, each blocked *clown* process times out after a preset amount of time. If this timeout occurs, the corresponding message is removed from the POP3 server (if it is still there), and a 451 error code is sent to the remote MTA to indicate a transient error, so that the latter can attempt to re-send the message at a later time.

3.4 MUA

The Mail User Agent is the software that the user usually interacts with when dealing with email. In our architecture, the MUA is responsible for simulating the behavior of a naïve user, opening every attachment and clicking on every link. Specifically, we use the popular Microsoft Outlook MUA [7] and the EZdetach [8] plug-in. EZdetach can extract and save Outlook attachments from messages, as well as run custom scripts on these attachments.

Outlook connects to the email-worm MTA through a secure IMAP connection and downloads suspicious messages from the server. As soon as a message is downloaded, attachments are extracted and run with administrator privileges. If these attachments contain malicious logic, RAD will detect anomalous behavior and notify *clown*.

4 Evaluation and Results

In this section we discuss the preliminary results of our proof-of-concept implementation. The results presented in this section are a coarse indication of overall system performance. The optimization of the system for deployment in large-scale organizations is the subject of future work.

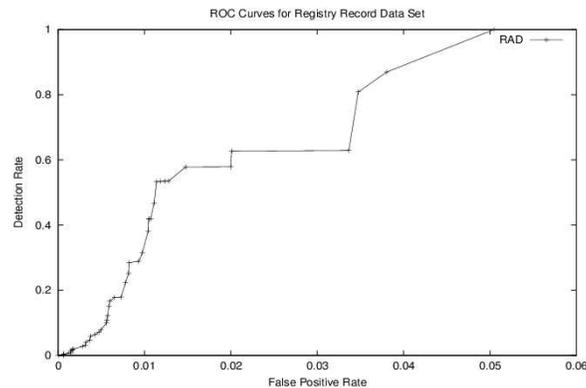


Fig. 3. Figure showing results of varying the threshold on the data set.

4.1 RAD

In order to evaluate the RAD system, we used the publicly available data set that can be found at <http://www.cs.columbia.edu/ids/rad>. The training data we used were collected over two days of “standard” user usage. Standard usage is defined as logging in, surfing the web, reading email, using a word processor, then logging off.

This simulated use of Windows produced a training data set (clean of attacks) consisting of 500,000 records. The attack data set (mixture of regular usage and embedded attacks) consisted of 300,000 records. The attack data set includes an array of publicly available attacks such as *Back Orifice*, *aimrecover*, *browlist*, *l0phtcrack*, *runattack*, *whackmole* and *setuptrojan*.

The natural way to evaluate the performance of an IDS is to compute the detection rate and the false positive rate. Detection rate is defined as the percentage of malicious programs that are correctly identified as anomalous by the system. The false positive rate is, in turn, defined as the percentage of normal programs diagnosed as malignant.

As illustrated in Figure 3, in order to achieve 100% detection rate, a false positive rate of 5% needs to be tolerated.

4.2 Timing

To test the efficacy of our system, we evaluated the performance of our system, in terms of detection latency, against real-world exploits. The tests were conducted on a PC with a 2GHz Intel P4 processor and 1GB of RAM, running windows XP Professional as the host operating system and Windows XP Professional as the guest OS under VMWare Workstation version 4.0.5.

The average time for downloading the message, detecting the attack and updating the MTA message queue was 28 seconds. Downloading the message and detecting using RAD all happen in sub-second times. The additional latency is imposed by the Microsoft Outlook MUA, as this is the minimum checking period allowed.

We can avoid this performance overhead by using a lighter-weight MUA (we implemented a simple IMAP client written in Python) but we would like to maintain the ability of detecting client-specific attacks. A hybrid approach can be used in cases where the extra overhead is not acceptable. In this scenario, the light-weight client would be used to check for suspicious attachments as a first step, and if none are found, continue with using a more widely used MUA.

Of further interest to the overall system performance is the cost of instantiating pristine VMWare images. To avoid the cost of moving around very large VMWare images which are in the range of 3GB, we set VMWare disks in non persistent mode and use the “repeatable resume” feature to quickly restart from a ready state. Restarting VMWare when using these features takes approximately 4 seconds.

| Message Avg | Virus Avg | Spam Avg | Delivered Avg |
|-------------|-----------|----------|---------------|
| 84966 | 4922 | 29726 | 50317 |

Table 1. Daily average email statistics collected from the Columbia University Computer Science department.

In order to get a rough estimate on the scalability of our system, we collected statistics from the mail server at Columbia University’s Computer Science department. Table 1 illustrates the daily average email characteristics for the time period of December 25th 2005 to January 24th 2005 as collected from the Sophos PureMessage email management solution. From the 50000 email messages received, approximately 8% contain attachments [9]. This translates to 4025 email attachments that need to be processed by our system. Given that processing time per email attachment is approximately 30 seconds, the system can process 3000 email attachments per day per VM. As VMWare ESX server can scale up to 80 powered-on virtual machines, our organization can rely on a single machine to handle daily email processing requirements. Obviously, these back-of-the-envelope calculations do not take into account the arrival rates and expected

delay per message but they serve as a rough estimate of what resources are required to deal with an enterprise environment.

5 Discussion

The two major goals of our architecture are scalability and reliability. Scalability will enable the use of the email worm detection architecture in a large-scale enterprise environment. To achieve this requirement, we need to minimize the rate of false positives in the host-based IDS, and speed up the detection stage on the virtual machine. Reducing the rate of false positives can be achieved by combining the RAD system with additional detectors such as the Windows Event Log data. This combination will allow for the use of data correlation algorithms that can be used, in turn, to produce more accurate behavior models. Reducing the time needed to detect malicious activity can be achieved by retrofitting MUAs to minimize the delay of checking and downloading messages.

Reliability will help our architecture in dealing with more complex issues such as targeted attacks against the system and encrypted email. One of the fundamental assumptions that our system makes is that the virtual machine can mimic the exact behavior of an operating system. If a worm can detect the presence of a virtual machine, it could potentially vary its behavior avoiding detection. The virtual machine that we choose for deployment in our system should successfully conceal its presence to the guest operating system as much as possible. In the absence of obvious clues from the VM, there are other techniques that an attacker can use (although not as reliable) to detect the presence of a virtual machine such as timing attacks *etc.* For this purpose, we can insert logic that identifies this sort of attempts.

The advent of end-to-end encryption mandates that our architecture should include a solution to address this problem. Storing all user keys on the mail server is not the best solution to this problem. However many organizations already require all email to be decryptable by them for legal reasons (*e.g.*, SEC regulations that cover all financial institutions in the US). Providing hooks to the MUAs in the virtual machine is one possible solution. This problem remains open for future consideration.

6 Related Work

Computer viruses have been studied extensively over the last several years. Cohen was the first to define and describe computer viruses in their present form. In [10], he gave a theoretical basis for the spread of computer viruses. The strong analogy between biological and computer viruses led Kephart *et al.* [11] to investigate the propagation of computer viruses based on epidemiological models. They extend the standard epidemiological model by placing it on a directed graph, and use a combination of analysis and simulation to study its behavior. They conclude that if the rate at which defense mechanisms detect and remove viruses is sufficiently high, relative to the rate at which viruses spread, they can prevent widespread virus propagation. [12] describes a filesystem layer designed specifically for efficient virus scanning and removal.

Also by Zou *et al.* [13], is the work that present an email worm model that takes into account the behavior of email users, specifically, email checking frequency and

the probability of opening an email attachment. They observe that the node degrees, as a logical network defined by email addresses, have heavy-tailed distributions. Their results indicate that email worms spread more quickly on a power law topology but are also easier to contain through immunization. In [14], the authors analyze network traffic traces collected for college campus environments and present an analysis of two major email-worm outbreaks, SoBig and MyDoom. Their work focuses on the effects of mass mailing worms on a single subnet. They show that both worms analyzed exhibit noticeable abnormalities in the traffic of the infected hosts.

The author of [15] proposes an automated email virus detection and control scheme using the attachment chain tracing (ACT) technique. This technique is based on epidemiological models that are used in infectious disease analysis and control. The author shows how these techniques can be used for detecting and immunizing an email virus.

One approach for detecting new email viruses was described in [16], which keeps track of email attachments as they are exchanged between users through a set of collaborating email servers that forward a subset of their data to a central data warehouse and correlation server. Only attachments with a high frequency of appearance are deemed suspicious; furthermore, the email exchange patterns among users are used to create models of normal behavior. Deviation from such behavior (*e.g.*, a user sending a particular attachment to a large number of other users at the same site, to which she has never sent email before) raises an alarm. Information about dangerous attachments can be sent to the email servers, which then filter these out. One interesting result is that their system only needs to be deployed to a small number of email servers, such that it can examine a minuscule amount of email traffic (relative to all email exchanged to the Internet) — they claim 0.1% — before they can determine virus outbreaks and be able to build good user behavior models.

MEF [17] is a UNIX mail filter that detects known and unknown malicious windows executables. By employing data-mining techniques on a database of known malicious executables, a generalized model is extracted that can, in turn, be used to detect future instances.

The work presented by Zou *et al.* [18] is probably the most closely related work. The authors present a feedback email worm defense system that uses a multi-step system for detecting new attacks. They also discuss the idea of using a honeypot system to detect outgoing traffic. Unfortunately, they provide no implementation details and do not address any of the apparent systems issues.

7 Conclusion

We have described a novel approach for scanning incoming email messages for *zero-day* worms and viruses. Briefly, we intercept all incoming messages, pre-scan them for suspicious content, and only deliver messages that are deemed safe. Instead of relying on a traditional signature-based approach, we employ a behavior approach where we actually “open” attachments inside an instrumented virtual machine looking for anomalous behavior.

We have implemented this architecture in a proof-of-concept implementation where we observe the behavior of different application on the Windows registry in real-time.

We show that we are able to detect the actions of all malicious software we tested at a false positive rate of 5%. Furthermore, we show that our implementation can be offloaded to any number of ancillary machines thus minimizing the computational overhead on the mail server.

Acknowledgements

We wish to thank Viktor Dukhovni for his invaluable help with Postfix.

References

1. : US-CERT Incident Note IN-2003-03: Sobig Worm . http://www.cert.org/incident_notes/IN-2003-03.html (2003)
2. : US-CERT Technical Cyber Security Alert TA04-028A: MyDoom Virus. <http://www.us-cert.gov/cas/techalerts/TA04-028A.html> (2004)
3. Spinellis, D.: Reliable identification of bounded-length viruses is NP-complete. *IEEE Transactions on Information Theory* **49** (2003) 280–284
4. Apap, F., Honig, A., Hershkop, S., Eskin, E., Stolfo, S.J.: Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. In: *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*. (2002)
5. : VMware. <http://www.vmware.com> (2004)
6. : Postfix. <http://www.postfix.org> (2004)
7. : Microsoft Outlook 2003. <http://office.microsoft.com/en-us/FX010857931033.aspx> (2004)
8. : EZdetach. <http://www.techhit.com/ezdetach/> (2004)
9. Stolfo, S.J., Li, W.J., Hershkop, S., Wang, K., Hu, C.W., Nimeskern, O.: Detecting Viral Propagations Using Email Behavior Profiles. In: *ACM TOIT 2005*. (2005)
10. Cohen, F.: *Computer Viruses: Theory and Practice*. *Computers & Security* **6** (1987) 22–35
11. Kephart, J.O.: A Biologically Inspired Immune System for Computers. In: *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press (1994) 130–139
12. Miretskiy, Y., Das, A., Wright, C.P., Zadok, E.: Avfs: An On-Access Anti-Virus File System. In: *Proceedings of the 13th USENIX Security Symposium*. (2004) 73–88
13. Zou, C.C., Towsley, D., Gong, W.: Email Worm Modeling and Defense. In: *Proceedings of the 3rd International Conference on Computer Communications and Networks (ICCCN)*. (2004)
14. Wong, C., Bielski, S., McCune, J.M., Wang, C.: A Study of Mass-Mailing Worms. In: *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*. (2004) 1–10
15. Xiong, J.: ACT: Attachment Chain Tracing Scheme for Email Virus Detection and Control. In: *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*. (2004) 11–22
16. Bhattacharyya, M., Schultz, M.G., Eskin, E., Hershkop, S., Stolfo, S.J.: MET: An Experimental System for Malicious Email Tracking. In: *Proceedings of the New Security Paradigms Workshop (NSPW)*. (2002) 1–12
17. Schultz, M.G., Eskin, E., Zadok, E., Bhattacharyya, M., Stolfo, S.J.: Mef: Malicious email filter - a unix mail filter that detects malicious windows executables. In: *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*. (2001)
18. Zou, C.C., Gong, W., Towsley, D.: Feedback Email Worm Defense System for Enterprise Networks. Technical Report TR-04-CSE-05, Univ. of Massachusetts, ECE Department (2004)