J Comput Virol (2005) 1: 4–12
DOI 10.1007/s11416-005-0003-8

ORIGINAL PAPER

Ibrahim K. El-Far · Richard Ford · Attila Ondi
Manan Pancholi

Suppressing the spread of email malcode using short-term message recall*

Received: 17 December 2004 / Accepted:14 March 2005 / Published online: 17 September 2005

Abstract Outbreaks of computer viruses and worms have established a pressing need for developing proactive antivirus solutions. A proactive antivirus solution is one that reliably and accurately detects novel malicious mobile code and one that either prevents damage or recovers systems from the damage that such code inflicts. Research has indicated that behavioral analysis, though provably imprecise, can feasibly predict whether novel behavior poses a threat. Nevertheless, even the most reliable detection methods can conceivably misclassify malicious code or deem it harmful only after substantial damage has taken place. The study of damage control and recovery mechanisms is, therefore, clearly essential to the development of better proactive systems. Earlier work has demonstrated that undoing the damage of malicious code is possible with an appropriate behavior monitoring and recording mechanism. However, it remains that even if a system is recovered, the virulent code may have already propagated to other systems, some of which may not be well-equipped in terms of proactive defenses. Curbing the propagation of undesired code once it has left the boundaries of a system is a hard problem and one that has not received much attention. This work focuses on a specific instance of this difficult problem: viruses and worms that spread by email. In this paper, we explore how advantageous it would be to have a short-term email undo mechanism whose purpose is to recall infected messages. Simulation results demonstrate that such ability can substantially curb the damage of email viruses on a global scale. The results are encouraging because they only assume technology that is either readily available or that is otherwise clearly practical today.

The authors would like to thank the Cisco Critical Infrastructure Assurance Group for their support in developing Hephaestus, and the Office of Naval Research (Award Number N00014-01-1-0862) for support in the ongoing development of Gatekeeper.

I.K. El-Far
Microsoft Corporation, 35/3307 Building 35, Microsoft Corporation, Redmond, Washington 98052, United States,
Tel: +1-425-706-6615
E-mail: ielfar@acm.org

R. Ford · A. Ondi · M. Pancholi
Florida Institute of Technology,150 W. University Blvd, Melbourne, Florida 32901-6975,
United States,
Tel: +1-321-674-7638,
E-mail: rford@se.fit.edu
E-mail: aondi@se.fit.edu
E-mail: mpancholi@se.fit.edu

1 Introduction

Two challenges to the development of useful antivirus solutions are the design of reliable, accurate virus detection strategies and the implementation of virus damage control and cure. In taking on these challenges, computer antivirus development has followed a technically simple model.

In following this model, members of the antivirus community monitor and report virus and worm activities. When a new virus demonstrates some risk to the computer user community, antivirus solution developers quickly analyze it in order to identify its characteristics and the damage it inflicts on systems and networks. They then build a detection mechanism that typically looks for some *specific* signature of that virus, and create procedures to cure infected systems. However, it should be noted that the design of most systems today makes reversing the damage of some viruses difficult if not altogether impossible after infection has occurred.

This process concludes with the distribution of the new detection and repair algorithms to users. In addition, the antivirus community attempts to raise user awareness of the potential threat and to improve their vigilance in spotting and avoiding it. This is sometimes called a reactive model because the community addresses a virus only after it finds it: protection is specific to a particular virus or worm. In the past, most threats have been slow to materialize and spread globally, and that has typically given the community enough time to respond. This, among other reasons, has contributed to the success of reactive models in mitigating the majority of past potential threats before they turn into epidemics [33].

While recognizing this success, the research community has long been skeptical of the reactive development model. In a paper on computer virus research [33], Steve White, of

the IBM Thomas J. Watson Research Center, identifies the development of proactive antivirus approaches as a key open problem for the research community. Development of heuristics represents a step toward detection of new viruses, but is provably incomplete [9]. The dependency on community vigilance, user awareness, and custom antivirus solution-update distribution is clearly risky. The evolution of modern systems, networks, programming frameworks, and computing resources have, predictably, contributed to the obsolescence of the reactive model in the face of novel viral, or otherwise exploitive, behavior.

Today's industry has a very pressing reason to recognize the reactive model is no longer sufficient: an increasing number of outbreaks of rapid-spreading, malicious, mobile code (MMC) has rendered available solutions all but useless in averting global epidemics. Many such outbreaks had serious ramifications for businesses and individuals, commanding substantial coverage in both technical and mainstream news media; [19], [20], [21], [13]. Code Red [2], [17], [36], Slammer [16], Blaster [18], and MyDoom [35] are some examples.

1.1 Proactive Solutions

The main objective of proactive antivirus solutions is to address previously unseen viruses. This is a difficult problem – indeed, some early research has focused on whether detecting novel viruses is even feasible. In a widely cited paper on computer virus theory [4], Fred Cohen formally proves that is not possible to create a static scanner that is completely accurate for all possible viruses. In a lesser-known paper [1] by Cohen's advisor, Leonard Adleman uses a different formalism and also arrives at the same result. The key takeaway from this theoretical work is that behavioral detection of viruses is probably the most viable option in dealing with previously unseen viruses. Another key result is that behavioral virus detection must be imprecise by nature. Another complicating factor is that over the years, the term virus, though precisely defined by Cohen and Adleman, has come to mean virtually any self replicating, propagating code that is a written with malicious intent, a concept which cannot be formalized.

The majority of detection work has so far focused on using a range of learning techniques to develop detection that is both reliable and accurate (A more reliable technique consistently detects more viruses. A more accurate technique consistently detects the same virus.). Many of the techniques that have been proposed to date focus on analyzing the behavior of suspicious processes. All learning techniques depend on how representative and rich a learning set is. They also depend on how much observable behavior they can afford to analyze before being forced to classify a particular object as benign or malicious.

Proactive detection is a very difficult problem, and the state of the art has a long way to go in terms of accomplishing practical accuracy (that is, reducing false positive and false negative rates). Unfortunately, the scope of this paper does not allow for further discussion of detection methods; the interested reader is referred to a paper by Jeremy Kolter and Marcus Maloof [12] for a more complete discussion of this isse. However, while detection has *received* substantial attention, the other challenge to proactive solution – damage control and recovery – has received considerably less scrutiny.

1.2 Damage Control and Recovery

A practical, proactive, and behavioral antivirus solution that does not require frequent user interaction must employ an effective strategy for reversing any damage that MMC inflicts on a system. In addition to the fact that fixing damage is important in and of itself, its study is worthwhile because it can have a positive effect on the quality of behavioral detection. A paper by Herbert Thompson and Richard Ford [9] suggests there is a strong relationship between the availability of damage reversal systems and the accuracy of behavioral virus detection, especially the false negative rate. In another paper [25], Dean Povey argues that a good recovery mechanism enables a retrospective, optimistic security paradigm where users may exceed their privileges when they need to and the recovery mechanism is applied when an infraction occurs.

A good deal of work done in the design of fault tolerant systems, preserving data integrity, and database backup systems is relevant to undoing the damage of a virus. Some work, however, specifically addresses recovering from malicious action. For example, a paper by Liu et al. [15] talks about how to reverse the effect of harmful transactions in databases considering the dependencies that legitimate transactions may have on the harmful ones. Another example is a paper by Tallis and Balzer [29] in which they discuss how to monitor the API of document processing applications in order to maintain a record of how documents are created and modified. The sequence of API calls used to create the document can be used to create an identical copy. This allows one to check whether a document has been altered by some unauthorized process or a different application.

In another conference paper [34], James Whittaker and Andres De Vivanco describe how they monitor the behavior of processes via the API calls they make. If a process is deemed malicious, this record of API calls is used to undo the changes made by this process. Andres De Vivanco, in [6], and Matt Wagner, [30], discuss different aspects of this in some technical detail. Despite this work, however, there are several problems that remain open in undoing the damage of viruses; these are detailed in the next section.

1.3 Objectives

The remainder of this paper examines the impact of implementing a short-term "undo" or "recall" mechanism for email exchange. The idea is that by implementing such a feature, behavioral virus detection could provide more accurate detection results while minimizing the spread of infection from one machine to another via "leaks" of malicious code during the detection process. The next section introduces aspects of the problem that are specific to our behavioral antivirus engine,

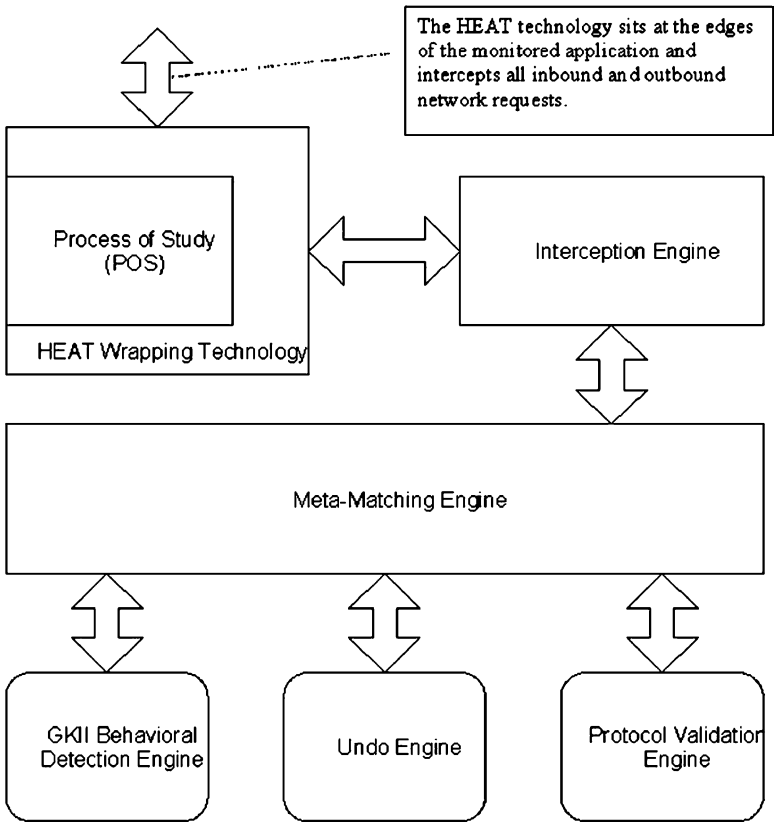

Fig. 1 Schematic diagram of the Gatekeeper architecture. The Protocol validation engine is shown simply as an example of the type of engines which can be included via the Gatekeeper API

Gatekeeper, and email undo. A presentation of simulation results that help evaluate the impact of email undo follows. This paper concludes with a discussion of the work in a larger context.

2 Short term email undo and behavioral detection

One of the challenges with behavioral detection is that by the time a sample is identified as hostile, changes and/or damage may have already occurred on the host machine. Thus, behavioral malware detection is not the same as behavioral malware *prevention*: MMC-related damage may have already occurred by the time a sample is classified as malicious in nature.

To this end, we have extensively researched behavioral virus detection coupled with the ability to undo certain changes on a machine. Our virus prevention solution, Gatekeeper, provides a robust investigation for such research. Architecturally, Gatekeeper has been described at length elsewhere [10]; however, a brief overview is provided here for clarity.

2.1 Design of Gatekeeper

The basic design of Gatekeeper is conceptually simple. Implemented for the Windows environment, Gatekeeper consists of three main components, a Monitoring Engine, one or more Matching Engines, and an Undo Engine.

The Monitoring Engine uses HEAT technology [34] to intercept calls a monitored program makes to the underlying operating system. These calls are passed to the Meta-matching engine, which pre-parses the data received and passes it on to one or more Matching Engines. One of the matching engines implements a robust "undo" function. The remainder provide classification between viral and non-viral samples. This arrangement is shown schematically in Figure 1. For the purposes of this paper, we will focus on the limitations and issues regarding undo.

As the Gatekeeper application is capable of monitoring all interactions of a monitored program with the system, the Undo engine simply has to "unwind" these interactions if the process is deemed to be harmful. This is a much less intrusive solution than a sandbox, as the monitored application interacts with the real system as opposed to a simulated or limited one.

Gatekeeper's Undo engine is a highly-granular backup-restore system that keeps track of the effects of individual computer processes on the protected systems. Traditional backup-restore solutions use system snapshots; they take periodic snapshots of a protected system, and, when it is deemed infected or otherwise unusable, it is rolled back to the most recent usable snapshot. This approach is effective in remov-

ing the effects of a malicious action but also results in the loss of benign and unrelated changes. For example, consider the case of a user working on a document in one window and receiving email in another. If the user is infected with a virus via email, a typical restore operation would entail losing all changes made on the machine since the time of the last system snapshot. Thus, undo operations that are global in nature, while helpful, can lead to additional loss of productivity.

In order to address these limitations, Gatekeeper focuses its undo operations on individual processes. As the monitoring engine reports API calls a process makes to the undo engine, the Undo engine records the API call, the action performed by the call, and backs up the data that will be affected by that action. For example, if the monitoring engine reports that a process made a call to the Windows kernel's DeleteFile, the undo engine notes the function, the file it is trying to delete, and backs up that file before the process is actually allowed to delete it. In this way, the monitored process is not "sandboxed" in that all actions actually occur on the machine. This helps reduce the impact of Gatekeeper on program behavior, and ensures that there is no loss of data reliability or stability due to the Gatekeeper process.

In implementing the Undo Engine, Gatekeeper makes use of a number of rules that explain how to undo specific actions. For example, every action that changes the content of a file can be undone by restoring a copy of that file taken right before the action is carried out; each modification to the registry can be undone by restoring the old registry key value. Similar rules can be devised for functions that modify security and sharing permissions, etc. Such rules are developed and set based on the various ways malicious code can make changes to the system.

The idea of Undo is not entirely new; Tallis and Balzer [29] explored a system very much like Gatekeeper's for implementing Undo operations. However, coupling Undo on a systemic level with behavioral detection is an innovation which provides for not only detection of new malware, but also its removal. In our tests, the Undo protection used within Gatekeeper has proven to be useful and reliable [10], and its efficacy for repairing viral payloads and infections has been demonstrated against Win32 samples contained on the WildList [23].

2.2 Limitations

The process of undoing changes on a machine caused by a specific process or process hierarchy is not trivial. There are several conditions that Gatekeeper's Undo engine is not able to adequately deal with. For example, consider two processes, B and M. Malicious process M modifies a file on the system. Before Gatekeeper determines that M is malicious, process B also modifies the file. Based upon this sequence, Gatekeeper will determine that the system is in a state where the changes by process M cannot be undone cleanly, as the changes by process B may have been made as a result of M's changes. Without specific knowledge of process B's functionality, it is not possible to ascertain the relationship of the changes.

Aside from such race conditions and unknown dependencies, Gatekeeper cannot undo actions that are not localized within the machine itself. For example, any network traffic that is generated by a viral process M cannot be easily undone. This issue, known as "leaky detection" is potentially serious, as some virus infections can spread from a protected machine.

In terms of network traffic, there is the issue of network traffic in general and the specific case of email. For network traffic in general, MMC generally attempts to exploit a particular vulnerability so sending out "undo" signals to hosts is not likely to be effective. However, the situation for email viruses and worms is very different. Here, the MMC does not immediately run on the remote machine. Instead, a well-defined protocol, SMTP [24], is used to transmit an infected object. This object does not generally use an exploit immediately on download; rather, it is only activated when the mail is delivered to the destination machine and that email is opened or previewed.

2.3 Email undo

Most users are familiar with the experience of sending an email and immediately wishing it could be recalled. Errors in addressing, reply-to addresses, content or simply accidentally hitting send have often caused users to wish for a robust email recall or undo function. However, there are many legal, social and technical issues associated with such a process.

For example, the ability to recall email generally is non-trivial from a technical perspective. Once email leaves the SMTP system and is delivered locally, it is not trivial to remove the mail from the client, even if it is as yet unopened. Similarly, recalling email is likely to increase the workload on a system that is already taxed by message volume.

Technical issues regarding recall are not even the most difficult part of the problem: if the offending email has already been delivered to the client, read, and acted upon, having the email later disappear without a trace from the mailbox could have legal and social ramifications. Thus, email recall in the general case is fraught with problems.

However, for behavioral virus detection, long-term email recall is not needed. Behavioral systems can generally detect viruses transmitted via email fairly easily; large numbers of emails do not usually "leak" from a Gatekeeper-protected system. Thus, in terms of virus prevention, the ability to undo email in the short term is almost as valuable as long-term undo.

We propose a system for undoing email designed for the explicit purpose of reducing the spread of email-aware viruses. This system would have short term (of the order of several seconds) undo ability, in order to minimize the burden on SMTP servers. Furthermore, once an email has left the control of the server (such as for an email downloaded via POP), undoing the email is considered to be no longer possible.

Such a system would be relatively lightweight to deploy, and not open to widespread abuse. Furthermore, as it would

not impact *read* emails, many of the social and legal objections to recall would become moot. Technologically, undo functionality would simply involve sending a second message with some cryptographically strong unique identifier that identified the email we wish to delete.

3 The simulation

In this section, we discuss how we simulate the spread of an email-transmitted virus, and demonstrate the benefit short-term email undo provides in the presence of "leaky detection". The section begins with a background on the simulation tool that we use and pointers to related tools and analysis work. It then describes the assumptions and set up of the simulation and presents the results and our observations.

3.1 Background

This paper makes use of Hephaestus [8][27], a simulation and modeling tool developed at the Florida Institute of Technology. Hephaestus is a discrete-time step Monte Carlo simulator that enjoys a number of modeling capabilities: it can model the spread of mobile malicious code through a large network; it can model host-based custom antivirus solutions; and it can model network-based solutions. During simulation, it is up to each node to exhibit interesting behavior: e.g. vulnerable to infection, infected, protected from infection. Finally, the simulator can handle several millions of nodes in practice within a reasonable time and using moderate computing resources.

Hephaestus is a member of the larger class of Monte Carlo simulators. Simulators are software systems developed to imitate the real world, and they have been used in science for both interpreting complex systems and predicting their behavior. Monte Carlo simulators choose the values of variables that are not deterministically predictable in a pseudo-random fashion. There are several examples of such simulators developed in relevant work that studies the spread of computer viruses and worms: Nicholas Weaver's simulator used to study Warhol worms [32]; Bruce Ediger's network worm simulator [7]; the Swiss Federal Institute of Technology's simulator used to study distributed denial of service attacks [26]; and Michael Liljenstam's simulator used to study such viruses as Code Red and SQL Slammer [14].

Other work in studying the spread of computer viruses follows an analytical approach rather than that of simulation. For example, Fred Cohen analyzed the transitive closure of information flow among computers to show that complete protection from a computer virus is not possible without imposing limitations on sharing of information or degree of connectivity [3] [4]. Alan Solomon developed an equation that predicts the rate of infection in previously uninfected machines will be proportional to the number of currently infected machines, to the number of currently uninfected machines, and to the probability of a given machine being infected [28]. Other examples include the work of Jeffrey Kephart and Steve White [11] and the work of Yang Wang and Chenxi Wang on epidemiological models of computer viruses [31].

In this work, the authors have chosen to use the predictive faculties of simulation to derive its results. Hephaestus addresses the needs of this work and has been a convenient choice for the authors. This convenience primarily stems from its ready availability and easy extensibility of its engine's source code. A good reference to the original design and implementation is Brian Shirey's Master's thesis [27]. We have extended that design in the course of this work to model arbitrary connections that reflect dynamic networks topologies such as those observed in email networks.

3.2 Assumptions

The simulation used in this paper models a typical mass mailing virus which uses the address book of an infected machine to find targets in a simulated network of 100,000 nodes to which it can spread. The simulation makes a number of assumptions:

1. All nodes are vulnerable to the simulated email virus.
2. All nodes behave identically.
3. The virus may or may not reach all the nodes, depending on the properties of the topology of connections.
4. In a typical setup, users read their email using a common email client on a common operating platform that manages their home or dedicated office computer. In the simulation, therefore, there is no distinction made among users, email servers, and email client machines: every user has a dedicated machine they use to check their emails and the emails are delivered to those machines instead of dedicated email servers.
5. All the contacts known to a given user are included in that user's address book.
6. There are no limits placed on the number of emails that a node can send in any given timestep. This is a reasonable simplifying assumption if timesteps in the simulation correspond to an order of several seconds or even a few minutes in real time.
7. Emails are delivered within one timestep of sending the message.
8. There are no lost emails.
9. There are no delayed emails.

3.3 Topology

To create the contents of the address books of email clients in our simulation we follow the topology described by Mark Newman in a paper on the spread of viruses over email networks [22]. The distribution of in-degrees across nodes follows a simple exponential function, and the distribution of out-degrees follows a stretched exponential with exponent 1/2. Every node has an address book that references other

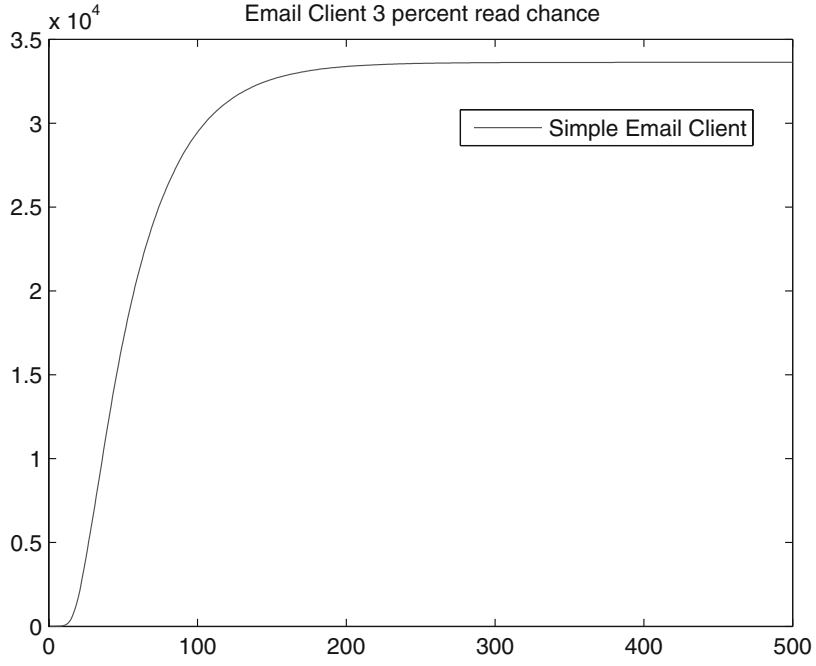

Fig. 2 A simple email client simulation with 100,000 nodes having no type of antivirus protection, executed over a period of 500 timesteps

nodes in the network with the number of entries and reciprocity of vertices simulated in accordance with the findings of Newman. The average number of entries in address books is approximately 12.

3.4 Behavior of nodes

The Hephaestus universe is made up of a number of nodes that together form the network within which the MMC spreads. In our experiments, the following node types and properties are used:

Simple Email Client Each node has an inbox of unread messages. The users have a chance of 3% of checking their emails in every timestep. This rate of message download is deliberately higher than one might predict, as it represents the worst-case scenario for our proposal. When a user checks their inbox and it contains one or more infected emails, the user's computer becomes infected and sends infected messages to every entry in the user's address book. This is a "worst case" scenario, as most email viruses do not spread just on preview or open.

Email client with Gatekeeper These nodes behave as described for the Simple Email Client, with the exception that they cannot send more infected messages than a set limit, indicating Gatekeeper's recognition and removal of the virus based upon its actions. Since Gatekeeper can only detect malicious behavior of individual applications, the node can be re-infected in subsequent timesteps.

Email client with email undo The nodes behaved as described for the Email Client with Gatekeeper with the exception that once Gatekeeper detects the virus, an Undo request is sent to every contact in the node's address book to which infected emails have been sent. Once an Undo Email is received, the infected message that has arrived in the previous timestep is removed if the message is still unread in the inbox. If the infected message has already been read, no action is taken in response to undo requests.

3.5 Setup

In every simulation we connected 100,000 nodes as described in the Topology section. To manage the typical statistical fluctuations from one run to the next, we ran the simulations ten times for each setup and averaged the results. In every case five nodes were selected randomly to be pre-infected with the virus.

3.6 Simulation with email clients

As shown in Figure 2, with no protection, the infection spreads rapidly among the nodes and approaches saturation at around 250 timesteps with a 3 percent chance that the user will read emails. The mean number of infected machines settles at around 34,000 infected nodes. Based upon the level of connectivity in our email network, this value seems to be accurate, and helps confirm the reliability of our simulator.

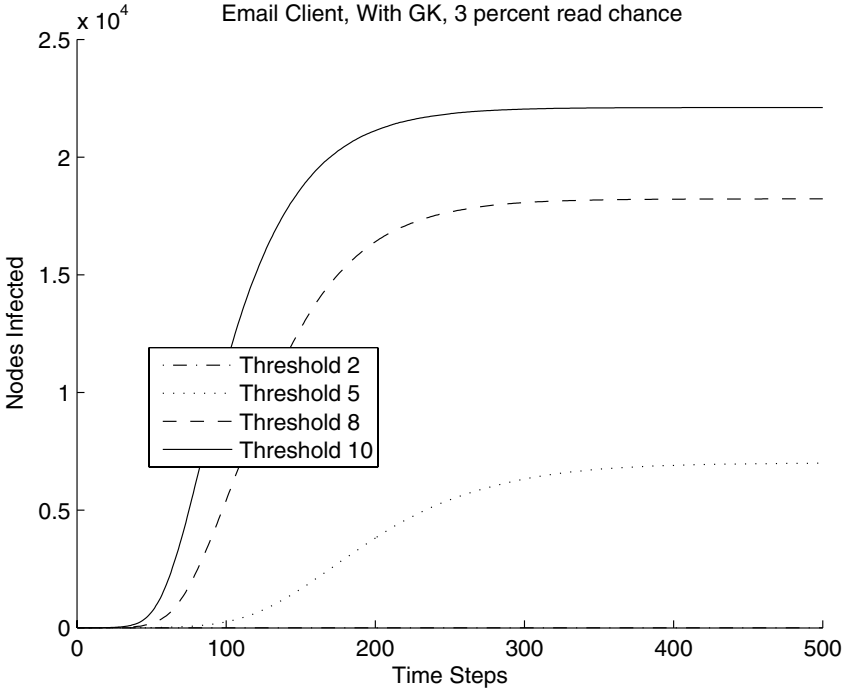

Fig. 3 A simple email client simulation with 100,000 nodes with Gatekeeper activated at thresholds of 2, 5, 8, and 10 and executed over a period of 500 timesteps

3.7 Simulation with gatekeeper

As shown in Figure 3, the results with Gatekeeper enabled show that it can significantly slow down the spread of email viruses, but only if it is able to identify the threat relatively quickly. As an example, with a threshold of 2, Gatekeeper is able to provide significantly more virus suppression than with a threshold of 10. This behavior is understandable: in a network, where the average number of entries in address books is 12, a threshold of 10 for Gatekeeper will let the virus send at most 10 infected emails (limited only by the address book), which means that it provides almost no limit to virus spread. When we compare the graphs in Figure 2 and Figure 3, it is clear that Gatekeeper can make a difference in the impact and spread of viruses, even though only a rather small percentage (26.6% for threshold 2, 18.3% for threshold 5, 13.8% for threshold 8, 11.7% for threshold 10) of machines have address books large enough to trigger detection

3.8 Simulation with email undo

The Email Undo results are even better – as expected – than the standalone Gatekeeper mechanism without undo. Short-term email undo reduces the spread by an order of magnitude even at the high 10 message detection rate. At a threshold of 2 and 5, Gatekeeper completely prevents an epidemic.

The explanation of the observed spread with a limit of 10 emails before detection is quite simple: email undo requests are only sent when Gatekeeper identifies the application as malicious. Therefore machines with small address books do not ever send enough email to trigger detection and therefore operate as though Gatekeeper was not running on them.

3.9 Observations

We have seen that machines with no protection are quite vulnerable to email viruses. Email-enabled MMC spreads quickly and effects a large part of the population. The results also show that Gatekeeper itself cannot prevent an epidemic of email-based viruses, but can introduce a significant delay to the onset of a full-fledged epidemic. This delay might be enough for the antivirus community to develop detection and removal algorithms to fight the virus before the infection grows too high. Undo results are even more promising, however. If Gatekeeper is quick in its recognition of the virus, Email Undo practically eliminates the infection cycle with very few nodes being infected.

4 Closing remarks

In this paper, we have examined the effect of "leaky" behavioral detection of email-aware viruses. Using our simulator, Hephaestus, we have modeled the spread of a worm that uses address books on infected machines in order to spread, and demonstrated that a behavioral approach which allows some infected emails to be sent before detection can still significantly delay an epidemic. Finally, we have demonstrated that

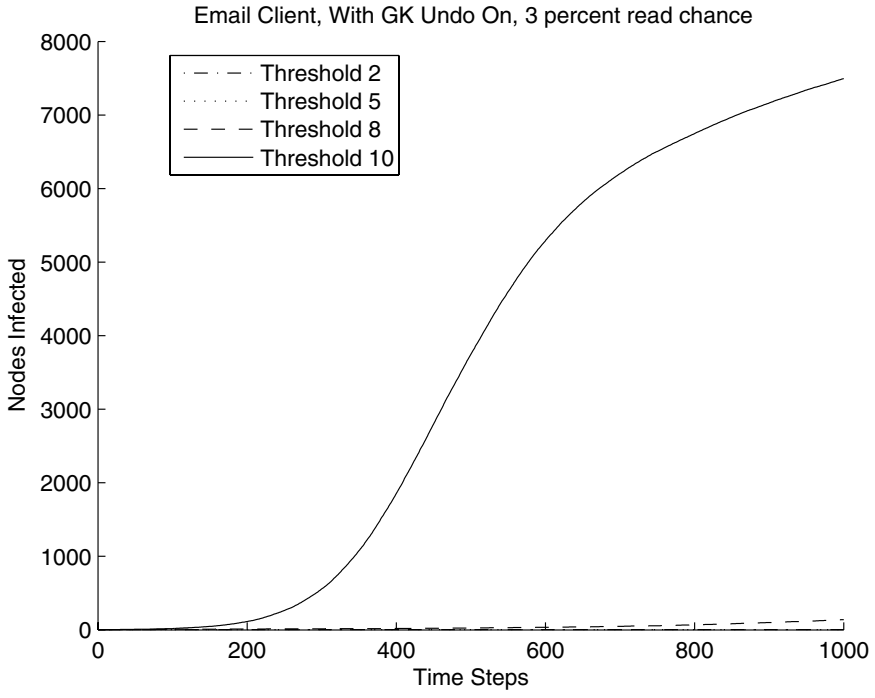

Fig. 4 A simple email client simulation with 100,000 nodes with Gatekeeper activated at thresholds of 2, 5, 8, and 10 with the undo feature turned on and simulation performed over 500 timesteps

even a short-term email undo function can greatly increase the efficacy of MMC suppression.

The need for such functionality is clear when considering behavioral detection. Late detection – that is, detection after a large number of behaviors have been observed – is more accurate than detection after only a few actions. Furthermore, there is no easy way for a behavioral system to safely buffer outbound email in order to gain a larger sample of program behavior.

Another possible solution to the problem on the Windows platform lies within MAPI the Microsoft Mail API [5]. If the operating system were to force all outbound email to use MAPI, and if MAPI had to be used asynchronously (that is, an application requests email service, and can send emails, but is informed asynchronously of success or failure), undo would be less critical, as outbound emails could be delayed until the behavioral system has gathered sufficient information to classify the sending program.

Overall, we have shown that the inclusion of a short-term email undo function in behaviorally-based anti-virus solutions would allow significant improvement in suppression rates. Furthermore, we argue that such a limited undo function overcomes the most serious objections to email recall in general. Finally, we suggest that the idea of "undoable" actions holds great promise for improvements to behavioral malcode suppression in general and urge authors and developers of systems to consider this fact when designing functionality.

References

1. Adleman LM (1990) An abstract theory of computer viruses. In: CRYPTO '88: Proceedings of the 8th Annual International Conference on Advances in Cryptology (Lecture Notes in Computer Science; Vol. 403), Springer-Verlag, pp 354–374
2. Berghel, H (2001) The code red worm. Communications of the ACM 44(12):15–19
3. Cohen, F (1986) Computer Viruses. Ph.d. thesis, University of Southern California
4. Cohen, F (1987) Computer viruses: Theory and experiments. Computers and Security 6:22–35
5. Microsoft Corporation. Messaging application programming interface (mapi), 2004. http://msdn.microsoft.com/library/default.asp?url=/library/en- us/e2k3/e2k3/ _techsel_tech_13.asp.
6. De Vivanco, A (2002) Comprehensive Non-Intrusive Protection with Data-Restoration: A Proactive Approach against Malicious Mobile Code. Master's thesis, Florida Institute of Technology
7. Ediger, B (2005) Simulating network worms. http://www.users.qwest.net/ eballen1/nws/.
8. Ford, R (2003) Microsoft, monopolies and migraines: the role of monoculture. Virus Bulletin Magazine
9. Ford R, Thompson HH (2004) Future of proactive virus detection. In: Proceedings of the EICAR Conference, Luxembourg
10. Ford R, Wagner ME, Michalske J (2004) Gatekeeper ii: New approaches to generic virus prevention. In: Proceedings of the International Virus Bulletin Conference Chicago, IL, USA
11. Kephart JO, White SR (1991) Directed-graph epidemiological models of computer viruses. In: Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy Oakland, California, USA IEEE pp 343–359
12. Kolter JZ, Maloof MA (2004) Learning to detect malicious executables in the wild. In: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining. USA, ACM Press pp 470–478

13. Legon J Tricky 'mydoom' e-mail worm spreading quickly, 27 January 2004. http://www.cnn.com/2004/TECH/internet/01/26/mydoom.worm/index.html.
14. Liljenstam M (2003) Ssf.app.worm: A network worm modeling package for ssfnet, http://www.crhc.uiuc.edu/mili/research/ssf/worm/.
15. Liu P, Ammann P, Jajodia S (2000) Rewriting histories: Recovering from malicious transactions. Distributed and Parallel databases 8(1):7–40
16. Moore D Paxson V, Savage S, Shannon C, Staniford S, Weaver N (2003) Inside the slammer worm. IEEE Security and Privacy 1(4):33–39
17. Moore D, Shannon C, Claffy K (2002) Code-red: a case study on the spread and victims of an internet worm. In: IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment Marseille, France pp 273–284
18. Morrison J (2004) Blaster revisited. Queue 2(4):34–43
19. Cable News Network. Code red computer worms targets white house, 2001. http://archives.cnn.com/2001/TECH/internet/07/20/code.red.worm/index. html.
20. Cable News Network. Microsoft offers virus bounty, 5 November 2003 2003. http://www.cnn.com/2003/TECH/biztech/11/05/microsoft.bounty/index.html.
21. Cable News Network. Slammer worm could pick up steam monday: New vulnerabilities might arise as businesses boot up, 27 January 2005 2003. http://www.cnn.com/2003/TECH/internet/01/26/internet.attack/index.html.
22. Newman MEJ, Forrest S, Balthrop J (2002) Email networks and the spread of computer viruses. Physics Review E (Statistical, Nonlinear, and Soft Matter Physics) 66(035101)
23. Wild List Organization. The wild list, 2005. http://www.wildlist.org/WildList.
24. Postel JB (1982) Simple mail transfer protocol (rfc821), http://www.ietf.org/rfc/rfc0821.txt.
25. Povey D (2000) Optimistic security: a new access control paradigm. In: NSPW '99: Proceedings of the 1999 workshop on New security paradigms Caledon Hills, Ontario, Canada, 2000. ACM Press pp 40–45
26. DDoSVax Project, 2004. http://www.tik.ee.ethz.ch/ ddosvax/.
27. Shirey CB (2004) Modeling the Spread and Prevention of Malicious Mobile Code Via Simulation. Master's thesis, Florida Institute of Technology
28. Solomon A (1990) Epidemiology and computer viruses. http://ftp.cerias.purdue.edu/pub/doc/viruses/epidemiology_and_viruses. txt.
29. Tallis M, Balzer R (2001) Document integrity through mediated interfaces. In: Proceedings of the second DARPA Information Survivability Conference and Exposition, Anaheim, CA, USA, 2001. IEEE pp 263–270
30. Wagner ME (2004) Behavior Oriented Detection of Malicious Code at Run-Time. Master's thesis, Florida Institute of Technology
31. Wang Y, Wang C (2003) Modeling the effects of timing parameters on virus propagation. In: WORM'03: Proceedings of the 2003 ACM workshop on Rapid Malcode, Washington, DC, USA. ACM Press pp 61–66
32. Weaver N, Warhol worms: The potential for very fast internet plagues, 2001. http://www.cs.berkeley.edu/ nweaver/warhol.html.
33. White SR (1998) Open problems in computer virus research. In: International Virus Bulletin Conference, Munich, Germany
34. Whittaker JA De Vivanco A (2002) Neutralizing windows-based malicious mobile code. In: SAC '02: Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain. ACM Press pp 242–246
35. Wong C, Bielski S, McCune JM, Wang C (2004) A study of mass-mailing worms. In: WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode, Washington DC, USA, ACM Press pp 1–10
36. Zou CC, Gong W, Towsley D (2002) Code red worm propagation modeling and analysis. In: CCS '02: Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, USA, ACM Press pp 138–147
37. Zou CC, Towsley D, Gong W (2003) Email virus propagation modeling and analysis. Technical Report TR-CSE-03-04, University of Massachusetts