Containing Hitlist-Based Worms with Polymorphic Signatures

Theodor Richardson* and Chin-Tser Huang†

*Dept. of Computer and Information Sciences
Towson University
trichardson@towson.edu

†Dept. of Computer Science and Engineering
University of South Carolina
huangct@cse.sc.edu

***Abstract*—Worms are a significant threat to network systems, both through resource consumption and malicious activity. This paper examines the spread of a class of hitlist-based worms that attempt to propagate by searching for address book files on the host system and using the host's mail program to spread to the addresses found. This threat becomes more severe when the worms are assumed to be polymorphic in nature – able to dynamically change their signature to elude capture. Because the method of propagation for these worms is predictable, it is possible to contain their spread through the use of honeytoken e-mail addresses in the client address book. Any e-mail received by the honeytoken address will be immediately recognized as malicious and can therefore be used to flag client machines as infected. This paper provides a complete description of a method to allow for better containment of this class of worms. The results of the proposed method are examined and compared to a previous method of capturing this type of worm.**

***Index Terms*—worm capture, worm containment, polymorphic worms, honeytokens**

I. Introduction

Worms are a significant threat to networks. They are self-contained executable programs capable of generating a large amount of traffic and performing malicious activity on a host system. They have the ability to copy themselves and propagate to other vulnerable hosts. Left unchecked, a worm will experience exponential growth as each copy of the worm continues to propagate. A worm can also contain a malicious payload that can disrupt, clog, or break an infected host.

Worms can typically be classified by their means of propagation. The two primary methods for worms to spread are by scanning and by hitlist. A scanning worm examines open ports across the network to determine possible hosts; when a connection can be made, the worm can spread without the action of a user. The scanning nature of these worms can often be used to detect when an attack is occurring and identify the machine responsible by the connection attempts [11][15][16].

The type of worm considered in this paper is the hitlist-based worm. A hitlist-based worm scans the host's system for address book files and other types of files typically containing e-mail addresses. These lists are then used to propagate the worm through the host's e-mail client. The difficulty in detecting these types of worms comes from their selection of existing addresses, meaning there is no randomness in their connection attempts. Typically, the only detectable activity for this type of worm is a disproportionately large volume of email being sent from a host, which could also be caused by benign means. This type of worm requires user interaction to infect the host, usually a user clicking on an attachment in an e-mail.

Detection of worms becomes a more difficult process when worms cannot be found by matching a consistent signature. The extraction of signatures becomes difficult when polymorphic worms are introduced. Such worms change their structure to evade detection. In [4], polymorphism is shown to be employed by the worm encrypting itself, reordering instructions, or other obfuscation techniques. As the malicious software changes, so does the signature. By switching registers or transposing code, a limited amount of variants are constructed that would increase the requirements for recorded signatures. However, a worm combining these techniques with a sophisticated encryption scheme could generate an immense number of different signatures. These worms can defy most standard signature-based malicious software detection systems, creating an even bigger threat once a host has been infected.

An encrypted worm cannot be executed without first being decoded. It must be rendered into an executable loaded into memory. However, Madou [7] explains techniques to break the code apart into different memory regions or to only decrypt small pieces when they are required. Even if this could be traced, the email client does not watch memory for executing programs. It is looking for the signatures of the email being transmitted. If this polymorphic worm is sending variants to each destination, multiple signatures would be required. The Storm worm used a number of techniques to generate over 40,000 known variants and transmitting thousands of emails per hour [1][17]. As the number of signatures increase, their use alone becomes increasingly impractical. Other techniques must therefore be applied to prevent the spread of polymorphic worms.

This paper presents a method for containing the spread of a hitlist-based polymorphic worm through a network by extending previous work on the use of a honeytoken e-mail account to detect malicious activity. The method proposed will allow the insertion of multiple honeytoken e-mail addresses into a client address book to increase the likelihood of a worm sending an infected email to a honeytoken as opposed to a legitimate address. The penalty system for an account sending to the honeytoken e-mail systems must also be revised to accommodate the lack of consistent signature for the polymorphic worm.

The remainder of this paper is organized as follows. Section II presents an overview of the work relevant to this paper, including methods for worm capture and containment and a summary of the previous honeytoken e-mail approach to containing hitlist-based worms. Section III provides a detailed description of the method and measures taken to prevent exploit and circumvention of the system. Section IV presents the experimental results from deploying the method and a comparison to the prior method. Section V presents a brief summary and conclusion.

II. Related Works

The typical method for detecting propagating worms is through a network intrusion detection system (NIDS), which will determine anomalous behavior when certain conditions are met. This can be either signature-based/misuse or anomaly detection. Signature-based NIDS detect attacks by comparison of network traffic against known attack signatures. Anomaly detection NIDS attempt to compare traffic against an expected baseline of 'normal' activity. However, the definition of this baseline is a complex problem [4]. The difficulty of applying these methods to a hitlist-based worm is that the only characteristic of the attack is an increase in the volume of email sent from a client, which could easily be caused by benign usage.

Similarly, methods have been developed to handle scanning worm behavior, typically by examining the number of scans versus the number of successful connections [15][16]. However, with hitlist-based worms, there are typically no failed attempts to connect since the worm will use the existing address book of the client to target potential hosts. Even signature-based comparisons on outgoing email are insufficient when attempting to contain a polymorphic worm that changes signature nonlinearly upon propagation.

This work makes use of the concept of honeytokens, intentionally inserted pieces of information that are enticing to a malicious user. These honeytokens have no legitimate usage within a system, so any traffic directed to a honeytoken immediately identifies a misbehaving client. For the purpose of this work, the honeytokens used are email addresses with no legitimate associated user in the system such that any email directed to the honeytoken address can be identified as malicious. All of these honeytoken addresses are directed to a running email daemon that will intercept the email and record the address and signature of the mail received and apply the appropriate penalty to the sender.

The work most closely related to this is the concept proposed for containing non-transformative worms by recording signatures received by a single dummy e-mail address [3]. Therefore, by scanning all incoming email for the signature(s) that are being penalized, the worm can be contained. However, even a single bit change in an email payload can cause a nonlinear change in the signature, making this system unsuitable for containing a polymorphic worm. Additionally, the capture rate can be improved (and the false positive rate reduced) by the use of a more complex system of assigning and maintaining the honeytoken email addresses, as demonstrated herein.

III. Proposed Method

There is currently no precise method for distinguishing automatically between legitimate and infected emails, and the most common method for determining whether an email is safe is by a comparison of its signature against the signature of a known malicious payload. This is typically insufficient, especially when considering the case where a benign email happens to have the same signature as the worm. In that case, a legitimate email would be blocked as a false positive for infection.

However, one proven method for definitively determining an email to be malicious is to use a honeytoken email account that under normal operation of the network would not receive any traffic. Therefore, any traffic destined to the honeytoken address can be assumed to be malicious. This method has been successfully applied to the capture of a non-transformative worm [3]. The honeytoken address is supported by a daemon that will receive the email and take appropriate action against the sender. The signature is also recorded and penalized in any future email while the signature is on the blacklist. However, this method is ineffective against polymorphic worms and results in a high number of false positive emails.

This concept can be modified and extended to allow for the containment of non-transformative worms and polymorphic worms with a much lower rate of false positives. The components of the system include the mail server, any number of clients, and a mail daemon to receive malicious email. The server is charged with maintaining a list of honeytoken addresses which will be inserted into each client address book prior to the send/receive of mail. It must also direct all mail destined for addresses in the list of honeytoken addresses to the daemon (which may be housed on the mail server itself). The only responsibility of the client machine is to house the honeytoken addresses in the address book (which may contain addresses outside of the mail server domain without effect on the utility of the method). This can be verified by the server prior to allowing the transfer of mail. The daemon is then responsible for penalizing misbehavior of any client sending an email destined to a honeytoken address.

A. Inserting the Honeytoken Email Addresses

The mail server is responsible for maintaining the list of honeytoken email addresses. These addresses must be generated in such a way that they do not overlap with legitimate clients on the network (assuming the server is responsible for all addresses in the network domain), but care must also be taken to avoid the addresses looking completely random since some worms have the potential to avoid addresses that look randomly generated. Following [27], the generation of an email address is done such that it includes a first initial, a proper name, and 2 to 3 digits to conform to the typical active username on a business/academic network.

It has also been demonstrated that evolving worms can filter addresses they have already seen a certain number of times *t*. Any instances of the address observed greater than *t* times will be ignored [3]. To prevent this from occurring, a more dynamic system for inserting honeytoken email addresses has been developed for this method. Each honeytoken address is therefore assigned a random time-to-live (up to a predefined value *t*) upon creation. This will mitigate the case where a worm will observe the address greater than *t* times and therefore ignore the address in propagation. The update of the current honeytoken addresses can be completed prior to the send/receive phase of the TCP/IP connection. The server must maintain the expired honeytoken addresses in its list for a time equivalent to the largest lifetime l_{max} possible for any address in order to capture any infected emails that were delayed in delivery either by the worm or network routing.

When any email is destined to a honeytoken address on the server's master list, it will be delivered to the daemon. All of the honeytoken addresses can be routed to the same daemon through standard forwarding techniques as long as the originating IP address and the signature remain intact. Some worms carry with them a lightweight SMTP server that is used on the infected client machine to contact external email servers to send mail. However, any mail destined to the honeytoken address will still pass through the server for delivery and get routed to the daemon.

Special care must be given to the distribution of the randomly generated honeytoken addresses in the address book to be sure that they have the maximum likelihood of being targeted by the worm. Worms generally attack target addresses in either a linear or random fashion. To account for both types of address book parsing, the list of honeytoken addresses must contain an address that comes alphabetically before any of the legitimate addresses in the address book as well as an address that comes alphabetically after the last legitimate address. It is possible that a client machine will have addresses in his/her contact list that are alphabetically before or after the honeytoken addresses, but these would be outside of the protected domain, and, following [18] can be ignored.

The rest of the addresses should be dispersed randomly throughout the address book such that the locations of honeytoken addresses cannot be predicted between updates. For example, in an address book A of *n* total legitimate and honeytoken entries, position A[0] and A[*n*-1] will be occupied by honeytoken addresses and any other honeytoken address may be structured alphabetically to occupy any position from A[1] to A[*n*-2]. Even if a worm arbitrarily bypasses a certain number of addresses within the address book, it still has a high probablility of encountering at least one of the honeytoken addresses. If a client address book is not organized alphabetically, measures must be taken to assert that the honeytoken addresses are inserted at the beginning, the end, and in suitably random positions throughout.

B. Send All

With the previous approach of capturing malicious email discussed in [3], a client would get penalized every time it chose to send to every address in its address book, because the address book includes the honeytoken addresses. To accommodate this function correctly, it is necessary for a client to offload this 'Send All' function to the mail server. By placing this on the server side, a client will not be penalized for a standard request since the server can easily filter out the honeytoken addresses.

However, the 'Send All' request must be suitably protected such that a worm cannot pass itself as a payload in the request and therefore bypass the honeytoken addresses completely. This can be accomplished by password-protecting the request to send to all entries in the address book. While this adds some nuisance to the client, it prevents bypass as well as eliminating the possibility of using the 'Send All' request to bottleneck the server by repeated requests without authentication credentials. To prevent replay of the authentication, a fresh token can be issued by the server at the time of the request for a 'Send All' such that this token must be returned as part of the request (along with the password). The message exchange between a client C and the server S for this occurs as follows:

1.) $C \rightarrow S$: Request to Send All $\| ID_C \| H(M)$
2.) $S \rightarrow C$: $H(K_S \| ID_C \| H(M))$
3.) $C \rightarrow S$: $ID_C \| E_{password}(H(K_S \| ID_C \| H(M))) \| M$

where M is the email message, ID_C is the identity of the client, H is an appropriate hash function such as SHA-1, E is a symmetric encryption using C's password as the encryption key, and K_S is the current authentication key for a 'Send All' request such that the request cannot be replayed with a different payload. In this way, a client cannot request a 'Send All' for one message payload and replace it with another message payload. The server response in Message 2 is a simple hash operation, which is relatively inexpensive even in quantity. This request is also assumed to happen during a send/receive, so a server cannot be clogged with messages requesting a 'Send All' outside of the standard mail upload and delivery time. If a particular session of send/receive is overpopulated with this type of request, the server can be limited by a threshold to process only a certain number per session (which would be a reasonable limitation based on

expected use of the 'Send All' function).

This method has two advantages that make it more secure. First, the user can be prompted to enter the password, so the password does not need to be saved on the client machine and the worm cannot find the password from the files. Second, the password is only used to encrypt the fresh token issued by the server, so the password itself does not need to be transmitted over the network.

C. Penalizing Misbehavior

When an email is delivered to the daemon, the client must immediately be penalized to prevent the worm from spreading to other clients within the network. Unlike the previous method for containing worms, it is not the specific signature that is most relevant in containing polymorphic worms but rather the client must be prevented from sending further email regardless of the signature of the email received. The signature received by the daemon, however, should still be penalized such that a non-transformative worm can still be captured and quarantined by this method.

This method can be managed by the use of threat indices. Each client will have an associated threat index as will each possible email signature. There is a predefined threshold T such that any client with a threat score greater than T will be blacklisted from delivering email. Similarly, any email with a signature matching a signature with threat greater than T will be quarantined. Whenever an email is received by the daemon, the following two steps are performed:

1.) The client threat index is increased by the maximum penalty P_{max} (where $P_{max} > T$)
2.) The signature threat is increased by a penalty P_s

Therefore, whenever an email is sent to a honeytoken address, the threat of the client will immediately exceed the threshold T and cause the user to be immediately blacklisted from sending further email. Additionally, any email with a signature whose threat is higher than T will be quarantined. Any email directed from the client who has sent an email to the honeytoken account will cause the threat score of the signature of the email to increment by 1. If the attack signature of the worm is non-transformative, the worm will then be detected and contained by the use of this threat score on email signatures. Periodically, all threat scores should be decremented to allow benign signatures and clients who accidentally email the honeytoken address to eventually recover privileges. The threshold T should be set higher than 1 such that a polymorphic worm will not blacklist each of the large number of signatures it generates. Users must also be blacklisted on the basis of IP or MAC address as opposed to their email address such that they cannot be blacklisted by an externally generated email intentionally targeting a honeytoken address. It is also only necessary to maintain threat scores for clients within the email domain of the mail server such that the system resources cannot be clogged by spamming the daemon with false account names.

IV. Experiments and Discussion

The method proposed herein is tested using vmware [28] on an isolated machine cluster. One virtual machine is used to simulate the email clients sending mail and another virtual machine is used to simulate the SMTP mail server along with the daemon. The virtual machines are connected by TCP/IP as in real mail systems. Reports are maintained by both machines and compiled to produce the evaluative results. A text dump of a Microsoft Outlook Express address book is used as the client address book for each client. It should be noted that an SMTP server that supports authentication is expected to prevent address spoofing attacks [12].

The client and server establish a TCP/IP connection for mail transfer. The client will provide authentication information (in this case username and password). After the client identity has been verified, the server will update the honeytoken addresses in the client address book. The client mail system must send an acknowledgement when this process is completed or the server cannot allow the upload of mail from the client machine. During the phase of send/receive mail, any requests uploaded from the client to 'Send All' must be handled by the server. This will involve the server responding with a token to uniquely identify the request and requesting the client to provide authentication details for the request; this will prevent an autonomous worm from replaying requests for the 'Send All' function to bypass the honeytoken addresses as well as preventing server clog by repeated requests. Upon completion of the session, any mail uploaded by the client machine is tested against the threat score of the client and the signatures of each email are tested against the threat score of each signature. If a client threat score is higher than the threshold T, all of the client's outgoing mail will be quarantined and each corresponding mail signature will be penalized by 1. If any of the mail signatures have a threat score higher than T, that mail message will be quarantined.

Any messages that are not quarantined are free to be delivered to recipients. Any mail destined to an address on the list of honeytoken addresses will be sent to the mail daemon, which will then record the IP address of the sender and the signature of the message. The sender will then receive the maximum threat increase P_{max} and the signature threat will be increased by P_s. The signature is penalized more heavily in this case because the email is known to be malicious.

Using this configuration, the goal of the test in vmware was to simulate usage of the email server on a network under attack to determine the accuracy and adequacy of the response provided. To configure the network, client load and values similar to those in [3] are used in order to provide a more thorough comparison to the previous use of this type of worm capture. For any of the attack scenarios, it is assumed that there are 21 infected clients and 25 legitimate clients with an overlap of 5 mutual addresses that will deliver both infected and legitimate emails.

For the timing of the simulation, the following parameters were used: 0-4 seconds between consecutive injections of emails from the clients (both infected and legitimate), 2.5 seconds between checks of the daemon to assign penalty, 5 seconds between gathering the state of the email server, and

an l_{max} of 3 seconds. Based on this specific experimentation, it was determined that a maximum of 10 honeytoken addresses in the address book (consisting of 40 entries) were sufficient to contain the spread of the worm; more than 10 addresses will deliver a decrease in the rate of false positive mail messages at the cost of server overhead in a diminishing return relationship. These times were chosen both to compare against the prior method experimentation and to examine the effectiveness of the method in extreme cases in which threat scores are diminished quickly, possibly without providing significant containment of the infected client.

The penalty assignment was structured such that a client will be penalized heavily for sending to a honeytoken address to assure capture of a polymorphic worm. The signature penalty is also significant to capture the signature of a non-transformative worm but the severity is less than the penalty incurred by the client such that a polymorphic worm will not cause the quarantine of a large number of signatures. Therefore, the client penalty P_{max} is set to 30 while the signature penalty P_s is set to 15.

The results of the experimentation are seen below in Tables I-IV. The capture rate in any attack is consistently 100%; in reality, it is reasonable to expect less than a perfect capture rate under real conditions but it should not fall significantly. Table I and II present the results from an attack of a non-transformative worm, the target for containment in the previous approach. Whereas the prior test conducted in [3] places the false infected rate at a maximum of 63.4% for a linear scanning worm and 21.6% for a random scanning worm, this method provides a much lower rate of 14.83% at maximum for any of the attacks. Each simulation was run for 240 seconds to observe the behavior over time to find a stabilization point of the expected percentage of email to the honeytoken address as well as the false infected rate; this point usually occurs when the rate does not incur any regular growth or decline in value. As seen in the tables below, the method proposed is sufficient to contain attacks by hitlist-based worms even when the worms are polymorphic in nature.

TABLE I
LINEAR SCANNING NON-TRANSFORMATIVE WORM ATTACK

Seconds	%To Honeytoken Address	%False Infected	%True Infected
15	0.90	6.38	100
30	0.44	6.47	100
45	0.29	6.85	100
60	0.22	7.00	100
75	0.17	7.30	100
90	0.14	7.05	100
105	0.12	7.21	100
120	0.11	7.23	100
135	0.09	7.69	100
150	0.08	8.36	100
165	0.08	9.14	100
180	0.07	9.80	100
195	0.06	10.97	100
210	0.06	10.84	100
225	0.05	10.36	100
240	0.05	10.19	100

TABLE II
RANDOM SCANNING NON-TRANSFORMATIVE WORM ATTACK

Seconds	%To Honeytoken Address	%False Infected	%True Infected
15	0.88	8.00	100
30	0.44	6.47	100
45	0.29	7.20	100
60	0.22	7.35	100
75	0.17	7.24	100
90	0.14	7.11	100
105	0.12	7.06	100
120	0.11	7.24	100
135	0.09	7.46	100
150	0.08	8.60	100
165	0.08	9.27	100
180	0.07	10.10	100
195	0.06	11.37	100
210	0.06	11.44	100
225	0.05	11.51	100
240	0.05	11.48	100

TABLE III
LINEAR SCANNING POLYMORPHIC WORM ATTACK

Seconds	%To Honeytoken Address	%False Infected	%True Infected
15	0.90	6.75	100
30	0.44	6.14	100
45	0.29	6.52	100
60	0.22	7.32	100
75	0.18	7.44	100
90	0.15	7.34	100
105	0.12	7.29	100
120	0.11	7.36	100
135	0.09	7.39	100
150	0.08	7.21	100
165	0.08	7.72	100
180	0.07	8.16	100
195	0.06	8.79	100
210	0.06	9.22	100
225	0.06	10.05	100
240	0.06	10.89	100

TABLE IV
RANDOM SCANNING POLYMORPHIC WORM ATTACK

Seconds	%To Honeytoken Address	%False Infected	%True Infected
15	0.89	6.85	100
30	0.44	7.88	100
45	0.29	8.03	100
60	0.22	7.90	100
75	0.17	7.65	100
90	0.14	7.55	100
105	0.12	7.32	100
120	0.11	8.00	100
135	0.09	8.72	100
150	0.08	9.47	100
165	0.08	10.69	100
180	0.07	12.50	100
195	0.06	13.99	100
210	0.06	14.83	100
225	0.06	14.48	100
240	0.06	14.33	100

V. Conclusion

This paper presents an extension of the method for using honeytoken email addresses to capture and contain hitlist-based worms. The focus of this work is the containment of polymorphic worms that are capable of generating a large number of signatures thereby making signature matching an inadequate measure for quarantine. However, through the use of multiple honeytoken addresses, it is likely that any hitlist-based worm will target one of the false addresses and identify the client as infected. This information can then be used to quarantine the client and prevent the worm from spreading. The use of signature penalties is also included such that non-transformative worms can be captured by signature matching. This work also extends the use of the 'Send All' ability within most mail clients such that a client will not be penalized for sending mail to all accounts in his/her address book.

The approach described herein is also extensible to detection of spamming accounts since most spamming behavior targets a subset of the full range of addresses in a domain. When the daemon receives a spam email from an external source, it can then block that sender by IP from delivering mail in the domain while the threat score is significantly high. Care must again be taken in this case to prevent address spoofing.

References

[1] E. Chien, "The Perfect Storm", http://www.symantec.com/enterprise/security_response/weblog/2007/01/the_perfect_storm.html

[2] A. Gupta and R. Sekar, "An Approach for Detecting Self-Propagating Email Using Anomaly Detection", Proceedings of RAID 2003, Pittsburgh, PA, Sep. 2003.

[3] C.-T. Huang, N. Johnson, J. Janies, A. Liu, "On Capturing and Containing E-mail Worms", Proceedings of the 25th IEEE International Performance Computing and Communications Conference (IPCCC), April 2006.

[4] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, G. Vigna, "Polymorphic Worm Detection Using Structural Information of Executables", http://www.auto.tuwien.ac.at/~chris/research/doc/raid05_polyworm.pdf

[5] C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks", Proceedings of CCS 2003, Washington, DC, Oct. 2003.

[6] W. E. Leland, M. S. Taqqu, W. Willinger, D. V. Wilson, "On the Self Similar Nature of Ethernet Traffic", Proceedings of SIGCOMM93, San Francisco, 1993.

[7] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. Sutter, K. Bosschere, "Software Protection through Dynamic Code Mutation", http://trappist.elis.ugent.be/~banckaer/documents/madou05software.pdf

[8] J. McHugh, "Intrusion and Intrusion detection", International Journal of Information Security, Volume 1 Issue 1 (2001), pp 14-35, 2001.

[9] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Slammer Worm Dissection: Inside the Slammer Worm", IEEE Security & Privacy, Vol. 1, No. 4, pp. 33-39, Jul. 2003.

[10] D. Moore, C. Shannon, and J. Brown, "Code-Red: a case study on the spread and victims of an internet worm", Proceedings of the Internet Measurement Workshop 2002, Marseille, France, Nov. 2002.

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

[12] J. Myers, "SMTP Service Extension for Authentication," RFC 2554, Mar. 1999.

[13] V. Paxson, "BRO: A System for Detecting Network Intruders in Real Time", Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, Jan. 1998.

[14] M. Roesch, "Snort – Lightweight Intrusion Detection for Networks", Proceedings of the USENIX LISA '99 Conference, November 1999.

[15] S. Sellke, N. B. Shroff, S. Bagchi, "Modeling and Automated Containment of Worms", to appear in Proceeding of International Conference on Dependable Systems and Networks (DSN), June 2005.

[16] S. Staniford, V. Paxson, N. Weaver, "How to 0wn the Internet in Your Spare Time", Proceedings of the 11th USENIX Security Symposium, 2002.

[17] J. Vijayan, "RSA: New Threats Could Make Traditional Antivirus Tools Ineffective", http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9010460&intsrc=article_more_bot

[18] N. Weaver, S. Staniford, V. Paxson, "Very Fast Containment of Scanning Worms", Proceedings of the 13th USENIX Security Symposium, 2004.

[19] M. M. Williamson, "Throttling Viruses: Restricting propagation to defeat malicious mobile code", Proceedings of 18th Annual Computer Security Applications Conference (ACSAC), December 2002.

[20] C. Wong, S. Bielski, J. M. McCune, C. Wang, "A Study of Mass-mailing Worms", Proceedings of the 2004 ACM workshop on Rapid malcode, Washington DC, October 2004.

[21] J. Xiong, "ACT: attachment chain tracing scheme for email virus detection and control", Proceedings of the 2004 ACM workshop on Rapid malcode, Washington DC, October 2004.

[22] C. C. Zou, W. Gong, D. Towsley, L. Gao, "The Monitoring and Early Detection of Internet Worms", to appear in IEEE/ACM Transactions on Networking, 2005.

[23] C. C. Zou, D. Towsley, W. Gong, "Email worm modeling and defense", Proceedings of the 13th International Conference on Computer Communications and Networks (ICCCN'04), October 2004.

[24] The Honeynet Project, http://www.honeynet.org/.

[25] Symantec Security Response, W32.Sobig.F@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html

[26] Symantec Security Response, W32.Mydoom.A@mm, http://securityresponse.symantec.com/avcenter/venc/data/w32.novarg.a@mm.html

[27] "Email Naming Standard for MS Exchange", http://intranet.uml.edu/it/email/documents/Email%20Naming%20Standard%20for%20MS%20Exchange.pdf

[28] vmware, http://www.vmware.com