On Capturing and Containing E-mail Worms

Chin-Tser Huang[*] Nathan L. Johnson[*] Jeff Janies[*] Alex X. Liu[†]
[*] *Department of Computer Science and Engineering*
University of South Carolina
{huangct, johnso66, janies}@cse.sc.edu
[†] *Department of Computer Sciences*
The University of Texas at Austin
alex@cs.utexas.edu

Abstract

Capturing an e-mail worm and containing its propagation as early as possible is desirable in order to provide better protection for the networks and hosts against severe damage that may be caused by the worm. In this paper, we propose a new approach that makes use of the propagating nature of e-mail worms. This approach inserts into each client's address book a dummy e-mail address that is not used by any registered user of the local domain, such that we can be confident that any e-mail destined to this dummy e-mail address is generated by an e-mail worm. The captured signatures can then be used to construct a user blacklist and a signature blacklist to contain the propagation of this e-mail worm. We also discuss how e-mail worms can attempt to bypass the dummy e-mail address, and propose countermeasures against these attempts. Our prototype implementation shows that this approach is easily deployable and is effective in containing e-mail worms.

1. Introduction

A worm is a piece of executable code that can infect other hosts in a network that are susceptible to infection due to the vulnerability of some software currently running on them. After successfully infecting some susceptible hosts, the worm further propagates itself to other uninfected, susceptible hosts from the infected hosts. Worms pose a significant threat to all kinds of Internet applications as they are capable of generating a huge amount of traffic to congest networks and disable database servers. A worm carrying malicious payload may cause further problems, including breaking or compromising infected hosts.

We can divide worms into different types according to their ways of propagation. Among all types of worms, two types are especially rampant and thus deserve more attention. The first type is called scanning worms. A scanning worm looks for susceptible hosts by scanning the target port(s) of other hosts over the network. The scanning worm can propagate itself to an uninfected, susceptible host without the interference of a user. Examples of scanning worms include Code-Red [6] and Slammer [5]. The second type is called e-mail worms, which is the subject of this paper. An e-mail worm needs the help of unwitting users to propagate to other hosts. Usually an e-mail worm is launched when a user unwittingly opens the attachment of an infected e-mail. When the worm program is executed, it searches the host for the address book file and other files of certain types that contain e-mail addresses, and then spreads by sending infected e-mails to the e-mail addresses it finds. Examples of e-mail worms include SoBig [20] and MyDoom [21].

It is desirable to capture an e-mail worm as early as possible, so that we can get clues about how to fix the vulnerability exploited by this e-mail worm. It is equally important to contain an e-mail worm as much as possible, so that we can quarantine infected user accounts and e-mails to keep them from further infecting other e-mail accounts. However, current network intrusion detection systems (NIDS) either are unable to detect the propagation of an e-mail worm in a timely fashion, or fail to distinguish between the e-mails generated by worms and normal e-mails although an anomalously large amount of e-mails has been detected. The inability of NIDS to detect worms often leaves network administrators helpless under a large-scale outbreak.

In this paper, we propose a new approach that captures and contains an e-mail worm once the worm infects any user account in the local domain. Our approach exploits the propagating nature of worms to differentiate e-mails generated by worms from normal e-mails. We insert a dummy e-mail address into the address book file stored in each user account on the local domain; this dummy e-mail address is guaranteed to be unused by any legitimate user of the local domain, and is changed periodically to beat the possible evolution of e-mail worms. A dedicated dummy client is deployed to observe any e-mail destined to this dummy e-mail address. Once the dummy client finds any e-mail delivered to this dummy e-mail address, we know that this e-mail is generated by an e-mail worm because no legitimate user uses this dummy e-mail address. Therefore, we not only detect that a user account is infected by an e-mail worm, but also capture the

signature of the e-mail worm in the message received by the dummy account. We show through our analysis and simulation that this approach is easy to deploy, and is effective in containing e-mail worms.

The remainder of this paper is organized as follows. In Section 2, we survey previous works that are related to worm detection, capture, and containment. In Section 3, we present the architecture of our approach. In Section 4, we discuss ways that an e-mail worm can attempt to bypass the dummy e-mail address, and propose countermeasures against these attempts. In Section 5, we discuss the maintenance of the two blacklists for containing captured e-mail worms. In Section 6, we discuss the results of our prototype implementation. Finally, we conclude our presentation and discuss future works in Section 7.

2. Related Works

The current practice of detecting self-propagating worms is through the use of NIDS. A NIDS is responsible for observing network traffic and will raise alarms or take corrective actions when an intrusion or an anomalous condition is detected. As shown by [4], there are two major approaches of NIDS: signature/misuse and anomaly detection. NIDSs that are based on signatures look for activities that match known attack signatures stored in their databases; examples of them include Snort [10] and BRO [9]. NIDSs that are based on anomaly detection look for deviations from pre-established statistical profiles of normal network traffic; examples include [1] and [2]. However, both approaches have limitations. The limitation of a signature-based approach is that it cannot detect novel intrusions. The limitation of an anomaly-based detection approach is that it is difficult to construct a stable statistic profile of the ever-changing network traffic, and hence false positive and false negative alarms are raised frequently. Moreover, it is difficult for the two approaches to adequately capture the signature of the worm.

There are studies targeting the detection, modeling, treatment, and containment of scanning worms [7][11][12][13][14][17]. However, these studies do not fit well into the case of e-mail worms because unlike scanning worms that propagate themselves as fast as the environment allows, e-mail worms need to wait for unwitting users to open infected e-mails. There are studies investigating the fundamentals of e-mail worms. Wong et al. [15] study the behavior and characteristics of e-mail worms by analyzing network traffic traces. Xiong [16] proposes a chain tracing scheme for detection and containment of e-mail worms. Zou et al. [18] model the behavior of e-mail users by considering e-mail checking time and the probability of opening attachments, and simulated e-mail worm propagation on different topologies.

In [7], the authors report that there are three potential strategies to mitigate the threat of worms: prevention, treatment, and containment. According to the analysis of the authors, containment is the most viable among the three strategies. Our approach serves as a foundation for efficiently containing the spread of worms, because the worm signature captured by our approach can be used to construct appropriate blacklists which can contain the worm in the local network and also be passed to the managers of other networks.

It is instructive to compare our approach with the Honeynet Project [19]. A Honeynet is a network of one or more honeypots. A honeypot is a host that is meant for an attacker to easily break into, and usually contains some deceptively precious resources or sensitive data so that it appears attractive to potential attackers. Once an attacker enters a honeypot, information on attacks and threats can be gained through logging and analyzing extensive interaction with the attacker. A Honeynet organizes several honeypots into a network so that more types of information regarding attackers can be collected. The approach of a honeypot or Honeynet is a non-intrusive approach, because honeypot hosts wait for visits by attackers and do not send packets into the network. By contrast, our approach, although also non-intrusive, is more active than Honeynet, because we induce an e-mail worm that has infected one user account in the local domain to spread itself to the dummy e-mail account, in order to distinguish infected e-mails from all other legitimate e-mails, and capture the worm's signature in the dummy account.

3. Capturing E-mail Worm with Dummy Address

The first and foremost problem encountered by an e-mail worm capturing mechanism is how to distinguish between legitimate and infected e-mails. The propagation of an e-mail worm usually has anomalous characteristics, for example transmission of an exorbitantly large amount of e-mails from infected hosts [3][15]. A widely adopted approach is to extract common signatures from these e-mails that can be used to distinguish future e-mails propagated by the same worm. However, it will take some time for a detection system to observe the anomalously large amount of e-mails. Furthermore, even if the observed amount of e-mails is anomalous, it may be benign.

This problem can be alleviated by the introduction of an e-mail account that receives only e-mails generated by worms. In this case, we are confident that an e-mail worm is captured once this account receives an e-mail, without

any labor of distinction between legitimate and infected e-mails. Such a solution can be realized as follows. The e-mail server generates a dummy e-mail address that is not used by any user in the local domain. When an e-mail client connects to the e-mail server, the server requests to insert the dummy e-mail address into the client user's address book.

The communication between an e-mail server and an e-mail client normally takes the following steps: the client opens a TCP connection to the server, the server optionally requests the client to provide username and password for authentication, and the client downloads e-mails from the server or uploads outgoing e-mails to the server. In our scheme, a step of dummy address update is added after user authentication and before mail transfer: the server retrieves the current dummy e-mail address and sends it to the client, and the client acknowledges the update after the dummy address is inserted into the user's address book. Note that for this scheme to be effective, no e-mail can be transferred until the dummy address is inserted into the client user's address book, because an infecting e-mail worm must propagate itself to the dummy address in order to be detected. The message sequence between the e-mail server and a client is shown in Figure 1.

An e-mail client daemon is set up for the dummy account. This daemon remains up permanently, so that if any e-mail destined to the dummy address is received at the e-mail server, this daemon can be notified and pick up the e-mail immediately. Thus, the capture of the worm occurs at the time of delivery.

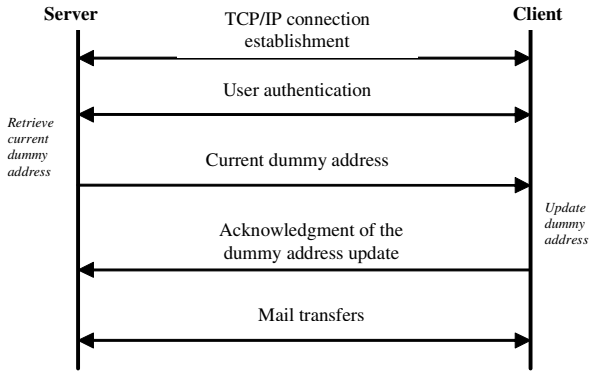

Figure 1 The communication between e-mail server and client has an additional step of dummy address update. This step falls between user authentication and mail transfer.

4. Attacks on the Dummy Address

Even if the dummy e-mail address is inserted into each client's address book, an e-mail worm can attempt to bypass the dummy address while infecting other e-mail addresses. In this section, we discuss some possible attacks aimed to bypass the dummy address, and propose countermeasures against these attacks.

An e-mail worm can arbitrarily bypass some e-mail addresses discovered in the address book of an infected account. If the dummy e-mail address is bypassed, then the worm will not send an infected e-mail to the dummy account and thus escape the capture. A countermeasure is to generate the dummy address in a random fashion, so that the dummy address is inserted into a random location in an alphabetically ordered address book. If the address book of a client is not alphabetically ordered, then the e-mail server requests the e-mail client to insert the dummy address into a random location in its address book. By doing such, the probability that an e-mail worm can avoid the dummy address by arbitrarily bypassing some e-mail addresses is small.

A more sophisticated e-mail worm can bypass those e-mail addresses that appear to be random combination of letters and numbers. A countermeasure to this attack is to make the dummy address indistinguishable from a normal e-mail address. A widely accepted e-mail account naming convention is to append some 2- or 3-digit number to a proper name [22]. This naming convention can be applied in the generation of the dummy address, by randomly choosing a name from a predefined list and appending a random 2- or 3-digit number to the chosen name. This random generation scheme makes it difficult for an e-mail worm to distinguish between normal e-mail addresses and the dummy address.

An even more sophisticated e-mail worm may "evolve" by storing the number of times it encounters the same e-mail address along its propagation, and bypass the e-mail addresses that it has seen more than *thr* times, where *thr* is a predetermined threshold. If the dummy address is inserted in all local domain users' address book, then there is a chance that this evolving e-mail worm will see the dummy address more than *thr* times during its propagation and will bypass the dummy address. A countermeasure to this attack is to change the dummy e-mail address periodically. Since an e-mail client always inserts the current dummy e-mail address into a user's address book before downloading or sending out e-mails for the user, it is guaranteed that the e-mail server and client are synchronized in the current dummy e-mail address.

An alternate method of attack requires the use of outside e-mail servers. In this method the infecting worm also carries a light weight SMTP server to be installed on

the victim system. This SMTP server can connect to other previously discovered or malicious e-mail servers. Therefore, an infected system has the ability to propagate further infection without the use of the client's designated e-mail server. In essence, a client of the protected e-mail server can become infected and use alternate servers to propagate infected e-mails. This approach has limited effect on circumventing our method of detection since an e-mail destined for a client of the protected server must pass through the protected server. If the infector attempts to communicate with the dummy address the source e-mail address and the e-mail signature are recorded. Even if the malicious SMTP server randomly generates an alternate source address for the infecting agent prior to attempting delivery of the mail, the mail's signature is added to a blacklist and further attempted infections with the same signature are prevented. This method does not prevent the spread of the worm outside the network, in this case, but clients using our e-mail server are still protected from infection. Similar to [13], clients outside the protected domain are beyond our concern.

5. Maintaining Blacklists for E-mail Worm Containment

E-mail worm signatures captured at the dummy account are used to construct blacklists for the purpose of containment. An e-mail that matches an entry in the blacklists is placed in a quarantine process and cannot be delivered until it is proven to be clean. In our scheme we maintain two blacklists: a User blacklist and a Signature blacklist. The maintenance of the two blacklists is summarized in Table 1.

5.1. User Blacklist

Every entry in the User blacklist includes the e-mail address, IP address of a blacklisted user, and a list of used signatures associated with this user. If an e-mail destined to the dummy address is received by the server and the sender of this e-mail is not blacklisted, then the e-mail address and IP address of this e-mail's sender is added to the User blacklist and the signature of this e-mail is added to the used signatures associated with this user. If an e-mail is sent by a blacklisted user, then the signature of the e-mail is added to the used signatures list associated with the blacklisted user if the signature is not yet blacklisted.

A user should not be blacklisted forever. A blacklisted user should be allowed to gain his/her rights back if it can be verified that the user's account is no longer infected. This is realized as follows. Each used signature is associated with a time-to-live value. Periodically each used signature is checked: if there is new occurrence of a used signature then its time-to-live value is set to its maximum value; otherwise its time-to-live is reduced by one. A used signature is removed when its time-to-live is down to 0. If no used signatures associated with a user are contained in the Signature blacklist, then the user is removed from the User blacklist.

5.2. Signature Blacklist

Every entry in the Signature blacklist includes a vector of attributes and their corresponding values, and a threat

Table 1 Two blacklists for containment of e-mail worms.

Blacklists of users and signatures		
Infected Lists	**Each entry contains**	**Actions to apply**
User	user (String) used signatures (List)	1. Add senders of e-mails destined to the dummy address to the User blacklist 2. Add new signatures of e-mails sent by a blacklisted user to the used signatures list associated with the user and set time-to-live to maximum 3. Periodically reduce the time-to-live of each user's used signatures and remove a used signature whose time-to-live is down to 0 4. Remove user from blacklist if none of its used signatures are blacklisted
Signature	signature (List) threat score (int)	1. Add new signature of an e-mail destined to dummy address to the Signature blacklist and initialize its threat score to *P* 2. Increase threat score of a matched signature by *P* if the received e-mail is destined to dummy address 3. Increase threat score of a matched signature by 1 if the received e-mail is not destined to dummy address 4. Decrease threat score of each signature by 1 periodically 5. Remove a signature from the list if its threat score is below 1

score. Possible signature attributes include subject line, packet size, hash of message payload, and source IP address. If a received e-mail is destined to the dummy address but its signature has not been blacklisted, then the signature of this e-mail is added to the Signature blacklist and its threat score is initialized to P, where P is a predefined penalty for a confirmed infected e-mail and must be larger than 1. If a received e-mail is destined to the dummy e-mail address and matches a blacklisted signature, then the threat score of the signature is increased by P. If a received e-mail is not destined to the dummy e-mail address but matches a blacklisted signature, then the threat score of the signature is increased by 1. Note that the two cases are treated differently, because an e-mail destined to the dummy address must be an infected one, but an e-mail not destined to the dummy address should be regarded as potentially infected.

A potentially infected signature should not be blacklisted forever, especially if it is blacklisted because its sender is blacklisted. This is realized by a redemption mechanism as follows: If t_{life} has passed, where t_{life} is an average period of time between the receipts of two legitimate e-mails, since last update of the threat score of a signature, then the threat score is decreased by 1. If a signature has a threat score below 1, then the signature is removed from the list.

Note that because of the redemption mechanism in both User and Signature blacklists, a legitimate user is able to redeem itself from the User blacklist quickly should it send an e-mail to the dummy address by mistake.

6. Implementation and Evaluation

We implement a prototype of our scheme with two Java Virtual Machine types: e-mail client and e-mail server. It is assumed that the server only supports IMAP connections, and it is integrated with an e-mail server. The server listens for connection attempts. The client functions much the same as a standard user system of an SMTP server, with one exception: it must validate its address book upon access. All communication between the two virtual machines is through TCP/IP connections, as would be the case in a real network.

The server machine maintains a complete user list, the dummy e-mail address, a blacklist for known infected e-mail addresses, a blacklist for known e-mail worm signatures, and the dummy address hash. Upon receiving a connection request the server checks the user name and password, then updates the client's address book with the current dummy e-mail address (for simplicity a text export of Outlook Express address book is used). After this process is complete the user's new e-mails are downloaded. An SMTP that supports authentication as defined in [8] should be used for the user to upload outgoing e-mails in order to thwart address spoofing attacks. This is similar to the operation of a normal e-mail server, but when an e-mail worm is found the source address is added to the user blacklist. No e-mail from a source on the user blacklist is allowed to be delivered, and no e-mail matching a known worm signature is allowed to be delivered. Instead these e-mails are placed in a quarantine box until further analysis is conducted to verify whether they are clean.

The signature blacklist is maintained by adding the signature of the message from a known infected e-mail address and increasing the threat score of the signature by one. This threat score is based on the signature and does not depend on the e-mail address of the infected client. After a predefined time period, t_{life}, the threat score of every signature is reduced by one. If at any time a signature's threat score is reduced below one, the signature is removed from the signature blacklist. When a blacklisted user uploads its outgoing e-mails a check is made to see if the blacklisted user has any known signatures remaining in the signature blacklist. If there are no signatures remaining in the signature blacklist that match to the blacklisted user's outgoing e-mails, then this user is removed from the user blacklist and is deemed clean. All e-mails from the user that were previously quarantined are marked as potentially clean.

The User blacklist is capable of redeeming innocent clients after a predetermined period of "good behavior". This mechanism allows redemption of clients that are "well-behaved" and prevents a worm from directly exploiting the Signature blacklist. For example, a malicious client attempting to cause more damage to the network mimics the signature of a "well-behaved" client, but intentionally causes the dummy client to trigger, thus denying service to a client [13]. For simplicity, during the parsing of the observed data results of our simulation tests, the determination of whether a client is legitimate or infected is based solely on its original status in the e-mail generators.

In addition to being able to detect and eliminate the original worms, our scheme detects signatures that are similar to what may have previously been a worm and quarantines the messages until the signature and sender are determined to be uninfected. Due to this method, a legitimate transfer may be tagged indirectly as infected if a matching signature is already blacklisted. Legitimate traffic is often sparse with many different signatures and redeems itself quickly if mistakenly tagged as infected. Our results demonstrate this by the infection of a legitimate sender with a legitimate message that match a signature of a previous worm that is still blacklisted. Due to the nature of this method, a user that is blacklisted indirectly without sending a message to the dummy e-mail address may only need to stay on the blacklist for a short

period of time before the user is considered clean. The duration of quarantine gives an administrator considerably more time to respond to the threat of an e-mail worm outbreak. Given enough time this system will correct itself and will not allow the containment scheme to perform a self-inflicted denial-of-service attack on the administrator's own e-mail server.

The client initiates TCP/IP connections with the server. After establishing a connection the client then transmits authentication information (username/password). After receiving confirmation of authorization the client receives the current dummy e-mail address. The previous dummy e-mail address in the address book is overwritten, and the client proceeds to download new messages.

The address book maintained by the client has only one requirement: the dummy e-mail address must be present. It is not assumed that only e-mail addresses under the domain of the simulated server can be in a client's address book. On the other hand, the server does not have to maintain the address books of all users of the system; the server just maintains the dummy address. The responsibility of the server is only to provide and ensure the presence and synchronization of the dummy address in the client's address book.

The goal of this design is to demonstrate the capability of this scheme to quickly detect e-mail worm propagation, protect clients from further infection, and capture an instance of the propagating worm. Two experiments are conducted with e-mail worms originating from inside and outside the protected network. The first experiment is a test of the capabilities of the system to detect the propagation of an e-mail worm that selects the next address to propagate (from the address book) in a linear fashion. The second experiment is to test the detection of an e-mail worm that selects the next address to propagate in a random fashion.

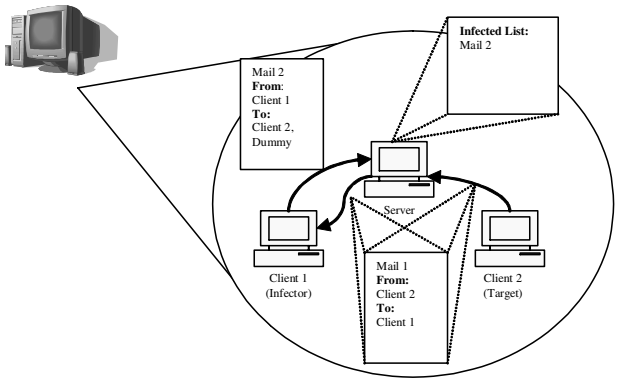

Figure 2 One computer simulates the interaction between three computers: the e-mail server and two clients.

The tests run on a single system with several components running simultaneously: target client, infector client, e-mail server, dummy daemon, and the data collector. An outline of this setup is shown in Figure 2. This setup is designed to be simple but without loss of generality. The target and infector clients generate traffic for the e-mail server, the dummy daemon periodically checks the dummy account, and the data collector dumps the current state of the e-mail server and the blacklists to a file for analysis.

The time parameters we use to conduct these tests are as follows: 0-4 seconds between two consecutive injections of e-mails from the infected and legitimate e-mail generators, 2.5 seconds between two consecutive checks of the dummy e-mail account, 5 seconds between gathering the current state of the e-mail server, 0.5 seconds for t_{life}. These time periods are very short, but they are intentionally chosen to demonstrate that this method can work in extreme cases where signatures may possibly be removed from the blacklist before they have a significant impact on containment. The value of the predefined penalty P needs to be large enough so that a confirmed worm signature will not be removed from the blacklist easily. We set P to 30 in the tests. A static number of e-mail addresses are used for the infected and legitimate e-mail generators. The specifics of the e-mail addresses can be seen in Tables 2 and 3. An infected address sends out infected e-mails, while a legitimate address sends out legitimate e-mails. The "mutual" addresses in Table 2 are the overlap between infected addresses and legitimate addresses, which means that they send out both infected and legitimate e-mails.

The results of the experiments are shown in Tables 4 and 5. In the case of an e-mail worm that selects the next address linearly, the system tags the infected clients with a success rate always higher than 95%. In the case of an e-mail worm that selects the next address randomly, the success rate is always higher than 72%. These results show that our scheme is effective at tagging infected clients. Note that the percentage of messages that are destined to the dummy address is very low, because only infected clients might send to the dummy address and not every infected client sends to the dummy address. However, the system is still able to tag nearly all the infected clients even though some infected clients do not hit the dummy address.

Table 2 Address book statistics for infected and legitimate clients

Number of Addresses	Total	Mutual
Infected	21	5
Legitimate	25	

Table 3 Sender and signature statistics for infected and legitimate clients.

	Infected	Common with Legitimate
Addresses	12	5
Signatures	12	5

Table 4 Simulation results in which the e-mail worm selects address in a linear fashion.

Sec.	Total messages	% to dummy	% false infected	% true infected
15	111	2.6	18.9	100
30	234	2.9	23.3	100
45	378	2.9	38.1	95.2
60	493	3.0	41.8	100
75	614	2.9	42.5	100
90	715	3.0	63.4	100
105	830	3.1	22.3	100

Table 5 Simulation results in which the e-mail worm selects address in a random fashion.

Sec.	Total messages	% to dummy	% false infected	% true infected
15	71	1.4	7.1	100
30	164	3.6	18.9	100
45	246	2.8	6.2	100
60	367	2.7	7.2	72.2
75	434	3.2	20.0	100
90	534	2.8	13.1	100
105	634	3.3	21.6	91.7

7. Concluding Remarks

In this paper, we present a novel approach to capture the signatures of e-mail worms and contain their propagation. An unused dummy e-mail address is set up on the e-mail server and inserted into every client's address book, such that we can be confident that any e-mail destined to this dummy e-mail address is generated by an e-mail worm. The signatures captured from an e-mail destined to the dummy address will then be added to a User blacklist and a Signature blacklist in order to contain the propagation of this e-mail worm. We have shown that this approach is easy to implement and deploy and does not affect normal network traffic in any way. If implementations of our approach are widely deployed, there is a good chance that e-mail worms can be detected and captured at an early stage of their propagation.

In the future, we would like to design a protocol that can be used among different implementations of our approach, so that e-mail servers can securely exchange collected information about e-mail worms with a centralized monitoring center or with peer e-mail servers, without being disrupted by impersonation attacks.

8. References

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

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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.

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

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

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