Containment of Fast Scanning Computer Network Worms

Muhammad Aminu Ahmad and Steve Woodhead

Faculty of Engineering and Science, University of Greenwich, UK
{m.ahmad,s.r.woodhead}@gre.ac.uk

Abstract This paper presents a mechanism for detecting and containing fast scanning computer network worms. The countermeasure mechanism, termed NEDAC, uses a behavioural detection technique that observes the absence of DNS resolution in newly initiated outgoing connections. Upon detection of abnormal behaviour by a host, based on the absence of DNS resolution, the detection system then invokes a data link containment system to block traffic from the host. The concept has been demonstrated using a developed prototype and tested in a virtualised network environment. An empirical analysis of network worm propagation has been conducted based on the characteristics of reported contemporary vulnerabilities to test the capabilities of the countermeasure mechanism. The results show that the developed mechanism is sensitive in detecting and blocking fast scanning worm infection at an early stage.

Keyword: worm detection, malware, cyber defence, network security

1 Introduction

Malicious software (malware) [1] is a generic term for any software that enters a computer system without the authorisation of the user to perform unwanted actions. Such software is a significant risk to the security of computer systems, with those connected to the Internet being at particular risk. Self-propagating malware (termed a worm) is a particular class of software which is rare, but particularly dangerous, because of its highly virulent nature. Fast scanning computer network worms are a particularly dangerous sub-class of such software.

The Internet has experienced a number of notable worm outbreaks (e.g. Slammer, Code Red and Witty [2]) that caused disruption of services and significant financial losses to government, transportation and other institutions [3]. However, the number of computer network worm outbreaks reduced significantly until the return of similar characteristics in the Stuxnet [4] outbreak that targeted industrial control systems in order to cause damage [5], which led to the release of other variants such as Duqu , Flame and Gauss for cyber espionage [4]. Vulnerabilities that can be exploited by a worm continue to be published by system vendors including the Microsoft RDP vulnerability (CVE-2012-0002) of 2012, and the ShellShock (CVE-2014-6271) and Drupal (CVE-2014-3704) vulnerabilities of 2014. The present threat of such an event therefore remains clear.

Previously reported research work used behavioural detection and suppression techniques at the host [6], local network and network perimeter [7] levels to counter the propagation of fast scanning computer network worms. However, there are limitations and shortcomings in the reported techniques. These limitations and shortcomings involve ineffectiveness in detecting worms, resource consumption, delay in deployment and detection, management overhead and computational complexity, and in most cases the techniques only slow, rather than stop worm infections [2],[8]. The previously reported research work can be categorized into signature-based and anomaly-based detection systems. The signature-based detection system maintains a database of signatures for previously known attacks and raises an alarm if a datagram in the network matches a signature in the database. Anomaly-based detection systems examine network traffic in order to build a profile of the normal behaviour and then raise an alarm for events that deviate from the normal profile. In contrast to signature-based systems, anomaly-based systems can detect new attacks. Anomaly-based detection systems look for deviations from the normal profile, based on the datagram-header information, payload information or both [9]. Datagram-header based anomaly detection systems use datagram header information to detect worm propagation. The focus of this paper is to develop an anomaly-based detection scheme using datagram-header information.

This paper presents a mechanism that uses two approaches for detecting and containing fast scanning computer network worms (abbreviated as fast scanning worms hereafter), which we have termed NEDAC (NEtwork and DAta link layer Countermeasure). NEDAC uses detection and containment techniques to defend against fast scanning worm attacks, which operate at the network level and data link level respectively. The detection part of the mechanism observes DNS activities to detect absence of DNS lookup in newly initiated outgoing connections. The containment part of the mechanism blocks outgoing traffic from a host that has been identified as infected, using data link access control.

The remainder of the paper is presented as follows. Section 2 presents related work on worm detection and containment systems. Section 3 presents an overview of wormable vulnerabilities. Section 4 presents the description of the developed countermeasure mechanism. Section 5 presents the experimental evaluation of the reported mechanism using a developed prototype. Section 6 concludes the paper and discusses possible future work.

2 Related Work

Significant research efforts have been devoted to the development of anomaly-based network intrusion detection systems, which have led to the existence of numerous approaches [8]. A number of these approaches [9] use datagram header information to identify the presence of computer network worms. Among these approaches are those that monitor source and destination IP addresses of datagrams, such as the work reported by Williamson [6]. Williamson [6] proposed a detection and suppression technique that uses the source and destination IP addresses of the host making a request to detect an attack. Whenever a request is made, the approach checks the newness of the host making the request by comparing the destination of the request to a short list of recently made connections. If the host is new it is then delayed, otherwise it will be processed as normal. However, many fast scanning worms (TCP-based) initiate connection requests to randomly-generated IP addresses, which results in a number of failed connections [2]. As a result, in addition to monitoring source and destination IP addresses, some approaches use the status of connection requests to detect worm behaviour such as the work of Jung et al. [10], Weaver et al. [11] and Rasheed et. al [12]. This technique uses the count of successful and failed connection attempts to determine the presence of worm scanning.

Furthermore, some detection approaches such as those reported by Gu et. al [13] and Mahoney and Chan [14] monitor source and destination ports and the Ethernet header fields. The work of Gu et al. [13] uses source and destination IP addresses and source and destination ports to detect fast scanning worms. This algorithm termed Destination Source Correlation (DSC), correlates incoming and outgoing traffic and keeps track of SYN datagrams and UDP traffic of the source and destination. Thus, if a host received a datagram on port i, and then starts sending datagrams destined for port i, it becomes a suspect. Then if the immediate outgoing scan rate for the suspect host deviates from a normal profile, the host is considered to be infected. Mahoney and Chan [14] developed the Packet Header Anomaly Detection (PHAD) technique, which learns the normal ranges of values for each datagram header field at the data link (Ethernet), network (IP), and transport/control layers (TCP, UDP, ICMP). PHAD uses the probability of rate anomalies in detection mode, based on the rate of anomalies observed during the training phase; the rarer the detected anomalies, the more likely they are to be hostile.

Another detection approach is to use DNS activities of hosts to detect worm propagation. Whyte et al. [7] and Shahzad and Woodhead [15] used DNS-based rate limiting to suppress fast scanning worms in an enterprise network. The observation was scanning worms often use numeric IP addresses instead of the qualified domain name of a system, which eliminates the need for a DNS query. In contrast, the vast majority of legitimate publicly available services are accessed through the use of DNS protocol; the network service that maps numeric IP addresses to corresponding alphanumeric names. Therefore the main idea behind this technique is that the absence of DNS resolution before a new connection is considered anomalous. This notion was first proposed by Ganger et al. [16], and if is implemented properly, it will impose severe limitations on worm traffic. This forces scanning worms to either probe DNS namespace or issue a DNS query for each IP address, which significantly reduces the speed of worm propagation [17]. The mechanism presented in this paper builds on the DNS-based detection scheme.

3 Wormable Vulnerability

According to Tidy et. al [5], a vulnerability is said to be wormable if it is network reachable, provides remote code execution, provides network access, and does not require human interaction once exploited.

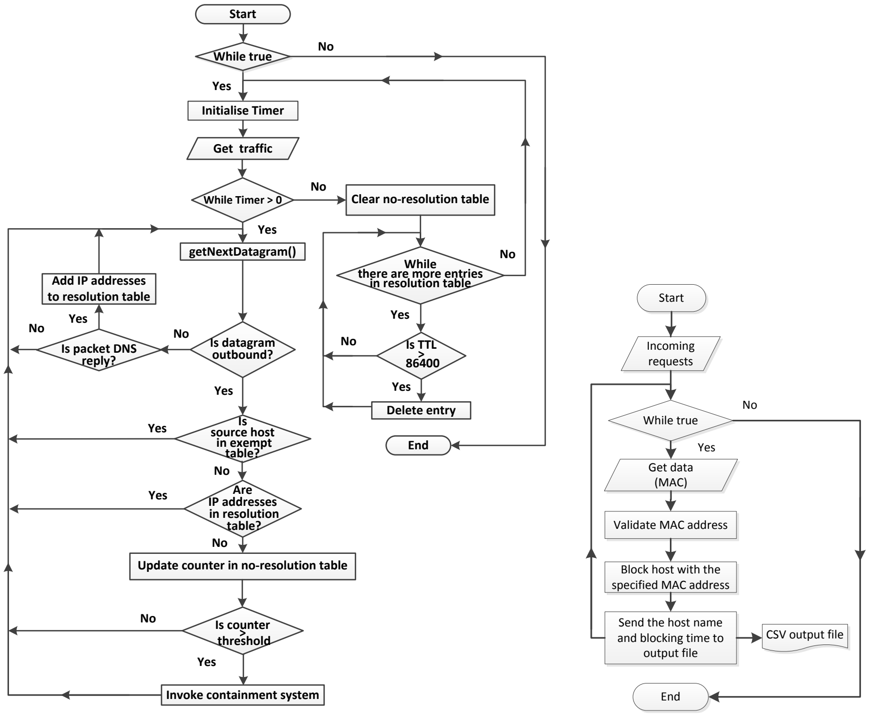
Individual vulnerabilities can be researched through a number of online sources that provide details of identified vulnerabilities such as the Common Vulnerabilities and Exposures (CVE) system [18]. The CVE system focuses on providing details for a range of vulnerabilities and keeps notes of whether a vulnerability is network reachable or requires human interaction if exploited. Additionally, Symantec Connect [19] provides working exploits for some vulnerabilities. These details provide the necessary information for assessing the wormability of many vulnerabilities. Some of the reported contemporary wormable vulnerabilities include Microsoft RDP (CVE-2012-0002) of 2012 and ShellShock (CVE-2014-6271) of 2014 [18].

4 Worm Countermeasure System

The proposed detection and containment mechanism uses DNS-based anomalies to detect the propagation of fast scanning worms in enterprise networks. Many fast scanning worms generate pseudo-random IPv4 addresses directly, without undertaking a DNS query. This behaviour obviates the need for DNS lookup, which is abnormal for the vast majority of legitimate publicly available services and is therefore a tell tale sign of scanning worm propagation [16]. Using a classification developed by Whyte et al. [7], the main focus of this paper is to detect worm propagation where the infection source is from local to remote and local to local using a detection system working at the network layer and a containment system working at the data link layer.

The NEDAC mechanism consists of two main sub-systems that work together to provide a countermeasure solution. The first system is the network layer detection system and the second system is the data link layer containment system, with a connection maintained between the two components to enable continuous data transmission. The detection system keeps track of all outgoing new TCP SYN and UDP datagrams by correlating them with a DNS resolution cache to determine the absence of DNS lookup. When a datagram is transmitted to a destination address without prior DNS lookup, the source IP address is maintained in a cache and its corresponding counter is incremented. The counter is incremented subsequently for every distinct datagram sent by a host without a prior DNS lookup. A threshold value, v, is set in order to assign a maximum number of distinct IP addresses a host can attempt to contact without a prior DNS lookup per time duration, t. Upon reaching the value, v, the detection system will mark the behaviour as worm propagation and therefore invokes the countermeasure by sending the MAC address of the source host to the containment system. The data link containment system listens on a TCP port for incoming connection from the detection system in order to block outgoing traffic from an infected host. Upon the receipt of a host MAC address from the detection system, the containment system will generate an access control update to block all datagrams originating from the specified host.

The design of the NEDAC mechanism is presented in Figure 1. Figure 1a shows the flow diagram of the network layer detection system and Figure 1b shows the flow diagram of the containment system.


Start
While true
No
Yes
Initialise Timer
Get traffic
While Timer > 0
No
Clear no-resolution table
Yes
getNextDatagram()
While there are more entries in resolution table
No
Add IP addresses to resolution table
Yes
No
Is packet DNS reply?
No
Is datagram outbound?
Yes
Is TTL > 86400
No
Yes
Delete entry
Yes
Is source host in exempt table?
End
No
Yes
Are IP addresses in resolution table?
No
Update counter in no-resolution table
No
Is counter > threshold
Yes
Invoke containment system
Start
Incoming requests
While true
No
Yes
Get data (MAC)
Validate MAC address
Block host with the specified MAC address
Send the host name and blocking time to output file
CSV output file
End

(a) Detection system

(b) Containment system

Figure 1: Flow diagram of the NEDAC mechanism

The detection system initialises a timer and then monitors TCP SYN and UDP datagrams. The detection system uses two caches, namely the resolution table and no-resolution table. Upon encountering a DNS response datagram, the algorithm records the host that made the DNS resolution and the resolved address in the resolution table. For outbound datagrams, the algorithm determines whether the source host is white-listed by checking the exempt table. The exempt table elements are a combination of IP addresses and port numbers that are exempt from the detection algorithm for known systems that legitimately communicate using IP addresses directly. If there is a miss, the algorithm determines whether there is a recent DNS query prior to sending the datagram by checking the resolution table. If there is a miss, the algorithm then increments the host's

counter in the no-resolution table and determines whether the entry exceeds a threshold. Upon exceeding a threshold, the system invokes the countermeasure solution by sending the source host MAC address to the containment system. After the expiration of the predefined timer, the system clears the entries in the no-resolution table, and then checks the resolution table in order to remove entries with expired TTL values.
The containment system receives MAC address of an infected host from the detection system and then blocks all traffic originating from the host. Finally, the system logs the host details and timestamp.

The NEDAC mechanism has been implemented as a software prototype using the C programming language. The C language provides low level programming support for network traffic using open source libraries such as *libpcap* [20] and *pjprojec*t [21], which facilitate traffic analysis.

5 Evaluation

This section presents an evaluation of the NEDAC mechanism. Firstly, analytical results of the number of susceptible hosts for the candidate contemporary vulnerabilities were presented. Then a description of the methodology used to evaluate NEDAC using developed worm outbreak scenarios was also presented. Finally, the section details the parameters used for the worm outbreak scenarios and the experimental results obtained.

The experiments reported in this paper use the Microsoft RDP (CVE-2012-0002) and ShellShock (CVE-2014-6271) contemporary vulnerabilities to develop potential worm outbreak scenarios for the evaluation of the NEDAC prototype. An initial challenge for the work was determining the values of the susceptible populations for these vulnerabilities. As a result, the CAIDA Internet Topology Data Kit (ITDK) [22] was used as a sample to determine the susceptible population values. The CAIDA ITDK includes passive traffic trace files for two Equinix backbones based in Chicago and San Jose in an anonymised format. Two trace files were collected from each centre. The four trace files, dated 20/03/2014, comprised approximately 47.85 million datagrams across a one minute period. The trace files were analysed and divided into two separate files containing datagrams originating from Windows hosts and from Linux hosts based on the reported IP header TTL of the datagram using Wireshark and Tshark [23]. The filters used to determine whether a datagram originated from a Windows or Linux host are ‘‘`ip.ttl>64 && ip.ttl<129`’’ and ‘‘`ip.ttl<65`’’ respectively.

Microsoft RDP protocol and mod_cgi are the main infection vectors for the Microsoft RDP and ShellShock vulnerabilities respectively. The mod_cgi is required by the popular host management tools Parallel Plesk and cPanel for certain modules, and so if it is possible to estimate the total number of hosts with these tools installed, this could act as a lower bound value for the number of Linux hosts with the module that could be susceptible to ShellShock. Such an estimate was developed by filtering datagrams with a destination TCP port equal to the management interface ports of the Plesk (8834) and cPanel (2083, 2082), compared to overall Linux hosts. However, RDP datagrams were filtered using TCP/UDP port 3389, compared to overall Windows hosts. The filtration of the datagrams was achieved using “`tshark -r <.pcap> -T fields -e ip.dst | sort | uniq | wc -l`”, where the “`sort`”, “`uniq`” and “`wc -l`” commands provide a count of the unique IP addresses that offer a particular service. The analysis further extrapolated the figures to determine a representative value of the entire IPv4 address space using $S_p = r * m * u_{ip}$, where S_p is the susceptible population, r is the ratio determined for each vulnerability from the dataset, m is the market share [24] of the target operating system, and u_{ip} is the routable IP address space; $3,673,309,759$ [25]. The market shares of Windows-based hosts and Linux-based hosts are 75% and 5.4% respectively of connected hosts on the Internet [24], therefore the average susceptible population values of RDP and Plesk/cPanel were estimated as 16.48 million and $42,533$ respectively.

To estimate the worm datagram sizes for experimentation, proof of concept exploits were collected from Symantec vulnerability database [19] for the Microsoft RDP and ShellShock vulnerabilities. The result of this estimation process was datagram sizes of 3.8 kilobytes and 2 kilobytes for RDP and ShellShock vulnerabilities respectively. These were used to configure the reported worm propagation experiments.

5.1 Experimental Methodology

The NEDAC prototype was deployed and tested in a virtualised network environment. The virtualised network environment comprises two personal computers with Intel Core i7 (12 virtual cores at 3.20 GHz) processor, 64GB of RAM and 2TB of hard disk storage capacity. The computers use VMware ESXi 5.5 [26] server to provide virtualization services, which enable the development of virtual networks on each computer in order to form a virtualised enterprise network. VMware ESXi has been chosen due to its strong performance in terms of the utilization of CPU, memory, disk I/O and network I/O [27]. The developed virtualised enterprise network of each computer comprises LANs with a DHCP server for IP address management, a DNS server for name resolution, an NTP server to provide a time synchronization service for the virtual hosts, a logging server to keep a record of worm infection activities and routers for internal routing services. Both internal and external routers have been implemented using the Quagga routing suite [28]. The detection system was installed on the gateway of each virtualised LAN and the containment system on the virtual switches of the virtual enterprise networks. Figure 2 depicts the logical and physical architecture of the virtualised environment.

Worm propagation behaviour was experimented using a worm daemon [29] that has been developed with the capabilities of facilitating a worm attack event using chosen worm characteristics. The worm daemon system consists of both client and server modules capable of sending and receiving UDP datagrams. The

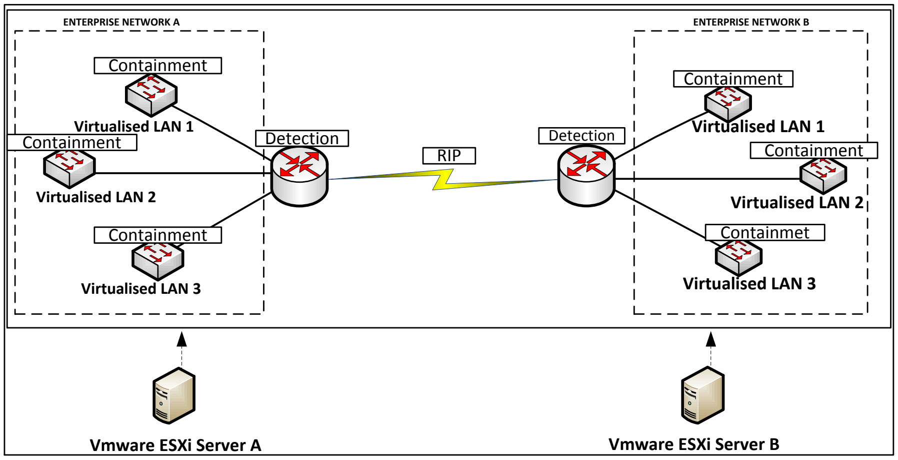


Figure 2: Virtual environment for countermeasure testing

client module is used to initiate a worm attack against the desired targets. Virtual hosts are made susceptible by running the server module, which listens on a specific UDP port and then, after receiving an "infection" datagram, continuously transmits "infectious" UDP datagrams. Upon infection, a susceptible host will send its time stamp and IP address information to the logging server for record management. The logging server has been configured with a logging daemon that keeps the details of infected host addresses and infection time. This process will continue until full infection is achieved based on the details recorded on the logging server. Finally, the experiment used Damn Small Linux (DSL) [30] as the operating system for the virtual machines. Furthermore, initiating a worm outbreak experiment involves creating the required number of virtual machines by cloning a base virtual machine that has been configured with the correct worm daemon. The virtual machines will then be powered to automatically synchronize their time with the NTP server, and then wait for inbound datagrams. The worm infection event is then initiated by sending a UDP datagram to one of the susceptible virtual machines in one of the virtualised LANs. A UDP-based worm has been chosen due to its higher rate of propagation compared to a TCP-based counterpart. UDP-based worms require no acknowledgement and cannot be detected by mechanisms that rely on number or state of failed connection attempts.

5.2 Experimental Parameters

The average susceptible population of hosts for each of the two candidate contemporary vulnerabilities and the size of routable IPv4 address space ($3,673,309,759$ [25]) were used to determine the number of susceptible hosts per million Internet hosts for each vulnerability using $P_m = \left[\left(\frac{S_p}{R_{ip}}\right) * 1,000,000\right]$, where, P_m denotes the value of susceptible hosts per million Internet hosts, S_p denotes the absolute number of hosts susceptible to the vulnerability and R_{ip} denotes the number of routable IPv4 addresses. The results were 4454 and 12 susceptible hosts per million for the RDP and ShellShock vulnerabilities respectively.

Another input value required by the worm daemon is the scan rate of the worm. The scan rate for each of the contemporary worm candidates has been determined using $\beta = \frac{U_{ip}}{S_p}$, where β denotes the scan rate. The resulting scan rates were 223 and 86364 "infectious" datagrams per second for RDP-based and ShellShock-based worms respectively.

5.3 RDP-based Worm Behaviour

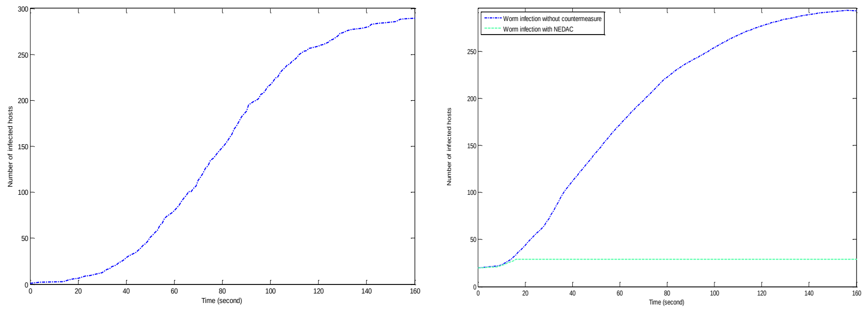
The RDP-based worm experiment was conducted using 4454 susceptible hosts per million in a single class B size network, and therefore contained $\left[2^{16} * \left(\frac{4454}{1000000}\right)\right] = 292$ susceptible hosts. The daemon was configured to listen on UDP port 3389 and then transmits UDP datagrams to port 3389 at a scan rate of 80 "infectious" datagrams per second, once "infected" using random seed. The scan rate was scaled down to 35% of the calculated value in order to avoid overloading server resources.

Five RDP-based worm experiments were conducted using one initially infected host without any countermeasure in place. Figure 3a shows the average result of the five experiments. The RDP-based experiment was repeated with NEDAC mechanism in place using a range of threshold values of 10, 20, 50, 100, 200, 400, 500 and 800 distinct IP addresses contacted without prior DNS lookup. NEDAC was configured to invoke the containment system if a threshold is exceeded within time duration of 10 seconds. The worm infection was detected and contained by the NEDAC mechanism with no further infection across the entire range of NEDAC experiments conducted.

The RDP-based worm experiment was also conducted with a hit-list [31] of 10 and 20 hosts in order to further evaluate the capability of the NEDAC mechanism. The hit-list behaviour was tested using threshold value of 800 and a time duration of 10 seconds. The worm propagation was also detected and contained with zero and nine further infections for the hit-list of 10 and 20 hosts respectively. Figure 3b shows the results of worm propagation using a hit-list of 20 hosts with and without the NEDAC mechanism.

5.4 ShellShock-based Worm Behaviour

The ShellShock-based worm experiment was conducted using 12 susceptible hosts per million in a single class A size network, and therefore contained $\left[2^{24} * \left(\frac{12}{1000000}\right)\right]= 203$ susceptible hosts. The daemon was configured to listen on UDP port 8080 and then transmits UDP datagrams to port 8080 at a scan rate of 86 "infectious" datagrams per second, once "infected" using random seed.



(a) No countermeasure in place (b) RDP-based with a hit-list of 20 hosts

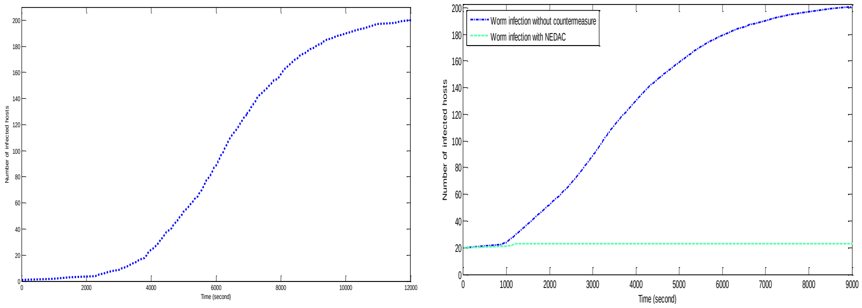
Figure 3: RDP-based worm propagation behaviour

The scan rate was scaled down by a factor of 1000 $\left(\frac{86{,}364}{1{,}000}\right) = 86$ in order to avoid overloading server resources.

As with the RDP experiments, five ShellShock-based worm experiments were conducted using one initially infected host without any countermeasure in place. Figure 4a shows the average result of the five experiments. The ShellShock-based experiment was repeated with NEDAC mechanism in place using a range of threshold values of 10,20, 50, 100, 200, 400, 500, 800 and 860 distinct IP addresses contacted without prior DNS lookup. The worm infection was detected and contained by the NEDAC mechanism with no further infection across the entire range of NEDAC experiments conducted.
The ShellShock-based worm infection experiment was repeated with a hit-list [31] of 10 and 20 hosts in order to further evaluate the capability of the NEDAC mechanism. The hit-list behaviour was tested using a threshold value of 860 and a time duration of 10 seconds. The worm propagation was detected and contained with zero and three further infections for the hit-list of 10 and 20 hosts respectively. Figure 4b shows the results of worm propagation using a hit-list of 20 hosts with and without the NEDAC mechanism.

6 Conclusion and Future Work

This paper has presented a mechanism, which comprises a DNS-based anomaly detection system and a data link layer containment system to counter the propagation of worms. The empirical results of the experiments conducted showed that the mechanism can detect and completely contain fast scanning worm including hit-list worm propagation scenario. This is due to the containment techniques employed in the data link layer that isolates a given infected host from the network and therefore ends the worm propagation.



(a) No countermeasure in place

(b) ShellShock-based worm with a hit-list of 20 hosts

Figure 4: ShellShock-based worm propagation behaviour

The experimental results for the RDP-based worm experiment without a countermeasure show attainment of 99% infection in 2 minutes. Thus using 224 scans per second for the RDP-based worm, the susceptible population of 294 hosts in a class B network could be infected in $\left[160\ sec * \left(\frac{35}{100}\right)\right] = 56$ seconds. However, despite the low population of susceptible hosts for the ShellShock-based worm compared to the RDP-based worm, the experimental results for ShellShock-based worm attained 99% infection in 200 minutes. The duration between detection and containment of an infected host was observed to be 1 second. Additionally, a hit-list was used to further evaluate the mechanism where 20 hosts were configured to transmit "infectious" datagrams at the same time. Nine further infections were observed after four seconds for RDP-based hit-list scanning and three further infections were observed after six seconds for ShellShock-based hit-list worm scanning. For RDP-based worm, the higher number of further infections can be explained due to large number of susceptible hosts compared to ShellShock-based worm. In both scenarios, further infections were observed due to the increased number of contacts made per second, i.e., $20 * 80 = 1600$ and $20 * 86 = 1720$ "infectious" datagrams for RDP-based worm and ShellShock-based worms respectively. In general, NEDAC has demonstrated effectiveness in detecting and containing fast scanning worms at early stage.

Furthermore, the speed of Internet connection available for an infected host and the worm datagram size determine how fast a worm can send datagrams. The Internet connection speed was estimated to be within the range10Mbps to 1000Mbps [32]. Using the Internet connection speed and a worm datagram size, the time T, required for a single worm instance, with size M (in bytes), to send datagram to a single IP address over a C megabits Internet connection can be determined using $T = \frac{M}{C} * 8$. Using 10Mbps Internet connection speed as a lower bound, the times required to transmit single datagram by RDP-based and ShelShock-based worms are 3 and 2 milliseconds respectively. Using 1000Mbps as an upper bound, the times required are 0.03 and 0.02 milliseconds for RDP-based and ShellSock-based worms respectively. Therefore the RDP-based worm can transmit 333 datagrams and $33,333$ datagrams using 10Mbps and 1000Mbps connections speeds per second respectively. Similarly, the ShellShock-based worm can transmit 500 datagrams and $50,000$ datagrams using 10Mbps and 1000Mbps connections speeds per second respectively. Thus, with these scan rates, NEDAC can detect and contain these contemporary worms in one second depending on the threshold value used, because a reasonable threshold value should not exceed 300 scans per second.

As future work, we plan to optimise the mechanism, particularly the detection scheme. It is believed that proper implementation of the DNS-based detection scheme will impose severe restriction on scanning worms. The mechanism will further be evaluated using a range of diverse worm scanning techniques such as stealthy scanning, local-preference scanning, topological scanning and evasive scanning. The effect of background traffic will also be tested to further evaluate the effectiveness of the mechanism and determine false alarms. The complexity of the detection system will be evaluated and then a comparative evaluation of the overall mechanism will be conducted.

References

1. J. Niemelä and P. Palomäki. Malware detection and application monitoring, November 2013.
2. P. Li, M. Salour, and X. Su. A survey of internet worm detection and containment. *Communications Surveys & Tutorials, IEEE*, 10(1):20 – 35, 2008.
3. C. Fosnock. Computer worms: past, present, and future., August 2005.
4. B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. The cousins of stuxnet: Duqu, flame, and gauss. *Future Internet*, 4(4):971 – 1003, 2012.
5. L. J Tidy, K. Shahzad, A. Muhammad, and S. Woodhead. An assessment of the contemporary threat posed by network worm malware. In *The Ninth Internation Conference on Systems and Networks Communications (ICSNC 2014)*, October 2014.
6. M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *In Computer Security Applications Conference, 2002, Proceedings, 18th Annual IEEE*, pages 61 – 68, 2002.
7. D Whyte, E. Kranakis, and P. C. Van Oorschot. Dns-based detection of scanning worms in an enterprise network. In *In NDSS*, February 2005.
8. V. Jyothsna, V. R. Prasad, and K. M. Prasad. A review of anomaly based intrusion detection systems. *International Journal of Computer Applications (0975 - 8887)*, 28(7):26 – 35, August 2011.
9. F. M. Cheema, A. Akram, and Z. Iqbal. Comparative evaluation of header vs. payload based network anomaly detectors. In *Proceedings of the World congress on Engineering*, volume 1, pages 1 – 5, July 2009.
10. Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 211–225. IEEE, 2004.
11. N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *In Proceedings of the 13th USENIX Security Symposium*, 2004.

12. M. M. Rasheed, N. M. Norwawi, O. Ghazali, and M. M. Kadhum. Intelligent failure connection algorithm for detecting internet worms. *International Journal of Computer Science and Network Security (IJCSNS)*, 9(5):280, 2009.
13. G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm detection, early warning and response based on local victim information. In *In Computer Security Applications Conference, 20th Annual IEEE*, pages 136 – 145, 2004.
14. M. Mahoney and P. K. Chan. Phad: Packet header anomaly detection for identifying hostile network traffic. Technical report, Florida Institute of Technology technical report CS200104, 2001.
15. Khurram Shahzad and Steve Woodhead. Towards automated distributed containment of zero-day network worms. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–7. IEEE, 2014.
16. G. R. Ganger, G. Economou, and S. M. Bielski. Self securing network interfaces: What, why and how? Technical report, CARNEGIE MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.
17. Cynthia Wong, Stan Bielski, Ahren Studer, and Chenxi Wang. Empirical analysis of rate limiting mechanisms. In *Recent Advances in Intrusion Detection*, pages 22–42. Springer, 2006.
18. CVE. *Common Vulnerabilities and Exposures*, 2014. [Online]. Accessed on 19th October 2014. Avalable: https://cve.mitre.org/.
19. S. Connect. *Vulnerabilities.* [Online]. Accessed on 12th November 2014. Available: http://www.securityfocus.com/.
20. Luis Martin Garcia. Programming with libpcap sniffing the network from our own application. *Hakin9-Computer Security Magazine*, pages 2–2008, 2008.
21. *PJPROJECT LIBRARY.* http://www.pjsip.org/.
22. *CAIDA, The Internet Topology Data Kit.* [Online]. Accessed on 11th November 2014. Avalable: http://www.caida.org/data/passive.
23. Gerald Combs. Tshark-the wireshark network analyser. *URL http://www. wireshark. org.*
24. *W3schools os statistics.* [Online]. Accessed on 12th November 2014. Avalable: http://www.w3schools.com.
25. M Cotton and L Vegoda. Special use ipv4 addresses. Technical report, BCP 153, RFC 5735, January, 2010.
26. Scott Lowe. *Mastering VMware vSphere 5.* John Wiley & Sons, 2011.
27. J. Hwang, S. Zeng, and T. Wood. A component based performance comparison of four hypervisors. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium*, pages 269 – 276, May 2013.
28. Kunihiro Ishiguro, T Takada, Y Ohara, AD Zinin, G Natapov, and A Mizutani. Quagga routing suite, 2007.
29. Khurram Shahzad and Steve Woodhead. A pseudo-worm daemon (pwd) for empirical analysis of zero-day network worms and countermeasure testing. In *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, pages 1–6. IEEE, 2014.
30. *Damn Small Linux.* [Online]. Accessed 19th October 2014. Available: http://www.damnsmalllinux.org/.
31. S. Staniford, P. Vern, and W. Nicholas. How to own the internet in your spare time. In *USENIX Security Symposium*, pages 149 – 167, August 2002.
32. *Net Index.* [Online]. Accessed 16 November 2014. Available: http://www.netindex.com/.