Network Worm Containment Using Markov Chain Approach

Akinwumi David
Information and Communication Application Centre,
Adekunle Ajasin University,
Akungba-Akoko, Ondo State, Nigeria

Alese Boniface Kayode & Oluwadare Samuel Adebayo*
Department of Computer Science,
The Federal University of Technology,
Akure, Ondo State, Nigeria
Email: *samoluwadare2013 {at} gmail.com

Abstract— In recent years, computer worms have emerged as one of the most potent threat to the security of many networked computing communities. The need for more reliable and efficient systems for worm containment has continued to be on the rise. Different systems for worm containment have been developed by different authors with attending strengths and weaknesses. Vigilante is a host based Intrusion Detection System (IDS) that detect worms by instrumenting vulnerable programs to analyse infection attempts. In this work, an improved Vigilante system that generates Self-Certifying Alerts (SCAs) using Markov Chain algorithm was developed. The algorithm is formulated such that upon detection, host generates Self-Certifying Alerts, which can be verified by any vulnerable host. Hosts receiving an SCA protect themselves by generating filters that block worm attack. The developed mechanism is implemented in Windows Vista environment using Visual Basic.Net programming language. Experimental results on different worms in the selected network demonstrate the system's ability to successfully detect and contain worms that are invoked into the network. A comparison of the results obtained with results of some other algorithms shows an overall good performance.

Keywords- ***Vigilante, Self-Certifying Alerts, Worm containment, Markov Chain***

I. INTRODUCTION

Computers worldwide have become increasingly interconnected particularly with the popularization of World Wide Web (www). The growing relevance of inter-network communication in our society today has made the Internet to become critically important to the viability of every sector of the national and global economy. However, an upsurge in the incidents of malicious codes in the form of computer viruses and worms witnessed in the communication system has posed a serious threat to the computing community. One class of such malicious code, known as worms, spreads itself without human intervention by using a scanning strategy to find vulnerable hosts to infect. Some of the famous examples of worms that have caused considerable damages are Code Red, SQL Slammer, and Blaster [1].

A worm is defined as self-propagating malicious program that exploits security vulnerabilities and flaws to propagate itself without requiring user action or human intervention. It performs self-replication by sending copies of their codes in network packets and ensuring the codes are executed by the computers that receive them [2]. Meanwhile, when a host or computers on network becomes a victim of its infection, it spreads further copies of the worm by exploiting low-level software defects [1]. The Slammer worm, for instance, attained probe and dissemination levels of as high as 26,000 scans for every second, a significantly high rate as compared to those that are realized for common viruses [3]. The following characteristics have been associated with the activities of worms:

a. Infection

Worms gain control of the execution of a remote program using one of these mechanisms: injecting new code into the program, injecting new control-flow edges into the program, and corrupting data used by the program [4].

b. Spreading

After infecting a computer, worms typically replicate itself to infect other computers, giving rise to a propagation process which has many similarities with the spread of human diseases [3]. The speed of propagation of worms depends on how fast infected computers can find new victims to infect [5].

c. Hiding

The following are the techniques used by worms to avoid being detected on the internet:

traffic shaping, polymorphism, and fingerprinting detectors

i. Traffic shaping: Worms usually have complete control over the network traffic generated by the computers they infect. By this means they can blend attack traffic with normal traffic, making it difficult to detect them by analyzing traffic patterns [6].

ii. Polymorphism: Another technique that worms can use to hide themselves is polymorphism. Polymorphic worms constantly change the content of their attack messages using techniques such as encryption and code obfuscation [7].

iii. Fingerprinting: Another technique that worms can use to avoid being detected is to try to identify if they are interacting

with a detector, before fully revealing their attack. This type of activity is referred to as fingerprinting the detector [8, 9].

The aggressive scanning traffic generated by the infected hosts usually cause network congestion, equipment failure, and blocking of physical facilities. For example, the Code Red worm version 2 released on July 19th, 2001 and over a period of less than 14 hours exploited buffer overflow vulnerability in the Microsoft Internet Information Service (MIIS) web servers, infected more than 359,000 machines. The cost of this epidemic, including subsequent strains of Code Red is estimated by a computer economist to be $2.6 billion [10]. This statistics revealed that Code Red was particularly virulent and the economic impact it provides indicates the magnitude of the damages that can be inflicted by such worms. Hence, there is a need to carefully characterize the spread of worms and develop an efficient strategy to contain them.

In this research publication, worm behaviours, methods of detecting them was examined, and an efficient and reliable Vigilante system for worm containment was developed using Markov Chain.

The major objectives of this research are to explore and bring to the fore the challenges of worm in a networked community and design and implement a Markov Chain based Vigilante system for worm containment that will also generate Self-Certifying Alerts (SCAs).

Markov chain, named after Andrey Markov, is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states. It is a random process characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property [10].

The term "Markov chain" is used to mean a Markov process which has a discrete (finite or countable) state-space. Usually a Markov chain is defined for a discrete set of times [12]. A discrete-time random process involves a system which is in a certain state at each step, with the state changing randomly between steps. Since the system changes randomly, it is generally impossible to predict with certainty the state of a Markov chain at a given point in the future. However, the statistical properties of the system's future can be predicted. In many applications, it is these statistical properties that are important in the propose system. The changes of state of the system are called transitions, and the probabilities associated with various state-changes are called transition probabilities. The set of all states and transition probabilities completely characterizes a Markov chain.

A Markov chain is a sequence of random variables X_1, X_2, X_3, ... with the Markov property, namely that, given the present state, the future and past states are independent. Formally,

$$\Pr(X_{n+1}=x \mid X_1= x_1, X_2= x_2, \ldots, X_n= x_n) = \Pr(X_{n+1}=x \mid X_n= x_n) \quad (1.1)$$

Markov chains are often described by a directed graph, where the edges are labeled by the probabilities of going from one state to the other states. A state *i* is called absorbing if it is impossible to leave this state. Therefore, the state *i* is absorbing if and only if

$$p_{ii} = 1 \text{ and } p_{ij} = 0 \text{ for } i \neq j.$$

II. LITERATURE REVIEW

To highlight worm behaviours and methods of detecting it, survey of some worm-related literature was carried out. The question that often arises when discussing worms is: what is the difference between a worm and a virus? Both are considered to be malware and can perform the same malicious actions. Viruses typically don't self-propagate, and rely on users to activate and transport the virus to a new destination. However, worms are generally self-propagating [5].

By definition, a worm is a self-propagating malicious program that exploits security vulnerabilities and does not require user action to propagate [2]. However, it has been shown that a computer worm is an extremely handy tool to perform a particular task in a distributed fashion or repetitively on several machines. A worm tries to hop on from one idle host to another carrying with it a sub-task in search of computing power to accomplish its tasks and return the results to the parent process that waits for the results on a different machine [13].

Worm's history of interest are discussed and presented as follows:

Creeper Worm was written by Bob Thomas and released in early 1970's, it was an experimental program to demonstrate the power of programming there was not malicious intent and the worms did not hide. Morris Worm was released in 1988. It located vulnerable hosts and accounts, exploited security holes on them to transfer a copy of the worm and finally ran the worm code. It penetrated remote systems by exploiting the vulnerabilities in either the finger daemon, send mail, or by guessing passwords of accounts and penetrate hosts that shared the same account [14.

Melissa Worm was a worm that caused wide spread damage to the internet and for the first time huge losses to everyone around the world. It caused over 400 million USD in damages across the globe and shutdown many organizations. It was written as a MACRO on Microsoft Word Document and this helped its widespread propagation. It was released in Mid March 1999 and was authored by David L. Smith [15].

ExploreZip took the concept of Melissa worm one step further. Melissa worm was not designed to reside on the system. ExploreZip was. The worm propagated via email, just like Melissa. Once the user opened the attachment, the worm would seem like a self extracting zip archive and then error out. Behind the scenes it would install itself on to the system and register itself in the Windows Registry. The worm would then stay dormant and do nothing. When the user reboots the system, the worm would get activated and mail a copy of itself to all the people in the address book of the user on the host. It would also delete all the C and C++ source files from the hard drive [16].

ILOVEYOU was written in VB Script and propagated as an attachment in the email with a message "ILOVEYOU".

When users opened this attachment, it would register itself onto the Windows Registry. This would activate the worm after every restart of the system. It would then, search all the drives connected to the host for all files with extensions *.JPG, *.JPEG, *.VBS, *.VBE, *.JS, *.JSE, *.CSS, *.WSH, *.SCT, *.DOC *.HTA, *.MP3, *.MP2 and rename them to .VBS [17].

Mydoom was the most notorious worms of all times with the highest damage of 22 billion USD. It propagated as a "Sending Failed" mail from the mail server and asked the user to click on the attachment to resent the mail. If the user opened the attachment, it would show that it's resending the mail and in parallel, installed the worm. The worm would then send a copy of itself to all the address in the address book and also copy itself to Peer-to-Peer shared drives. The worm also opened a back door for the hacker to get back anytime [5].

Code Red used a buffer overflow vulnerability to infect Microsoft IIS web servers. It would scan and infect other hosts until the 20th day of the month and then send a DoS attack to the whitehouse website until the 28th and then become dormant for the remainder of the month. Code Red was a memory-resident worm, so it did not persist across reboots [3]. Code Red II used the same buffer overflow vulnerability as Code Red, but was otherwise completely different. It first determined if Code Red II was already installed, and if not it installed a backdoor, went dormant for a day and then rebooted the machine. It then began to spread. Installing the backdoor allowed a remote user to execute arbitrary code at a later date. Slammer was the fastest computer worm in history [3]. It infected more than 90 percent of vulnerable hosts within 10 minutes [18]. Slammer exploited a buffer overflow vulnerability in computers on the Internet running Microsoft's SQL Server or MSDE 2000. It was a worm that picked its next victim randomly.

If a worm author collects a hit-list of a few thousand potentially vulnerable machines, ideally ones with good network connections. When released onto a machine on this hit-list, the worm begins infecting hosts on the list. When it infects a machine, it divides the hit-list into half, communicating one half to the recipient worm and keeping the other half. Such a worm is called a Warhol Worm and such a scanning technique hit-list scanning [5].

Nimda copied itself to network drives, shared the computer's folders, and created a guest account with Administrator privileges. It attached itself to explorer.exe to hide itself. It emailed itself to email addresses in the user's contact list. It was self-modifying, so hashes wouldn't identify it [19].

Slapper spread by exploiting vulnerability in the OpenSSL implementation used by the Apache web server. It scanned for targets by randomly choosing a network, and then sequentially scanning each IP in that network[20].

Blaster exploited a buffer overflow vulnerability in the RPC implementations of Windows XP and 2000. Blaster was uploaded to the target in two stages. The first stage transmitted itself via the RPC vulnerability, which then retrieved and executed the rest of the worm. Blaster was designed to send a SYN flood to windowsupdate.com on certain dates [21].

III. DESIGN OF THE PROPOSED VIGILANTE SYSTEM

a. *Method and materials*

The major focus of this work and publication is to develop an enhanced Vigilante system for worm containment using Markov Chain algorithm. The rationale behind the proposed system is to provide a system that guarantees a reliable and efficient worm containment system in a network.

b. Detecting and Containing a Worm

The first step towards containing the outbreak of an unknown worm is to detect it. To detect a worm outbreak, a defensive program must be instrumented to analyze its infection attempts. The detector must be duly informed about the outbreak and must be able to generate automatic self certified alerts (SCA), i.e. a security alerts that can be verified by the computers that receive them. The hosts receiving an SCA, protects themselves by generating filters that block worm attack, thus cooperating to contain an outbreak.

The self-certifying alert mechanism allows detection in Vigilante to be very dynamic, for the following reasons.

i. any host to independently decide to become a detector at any time, because detectors are not trusted. This makes it harder for an attacker to know exactly where detectors are deployed, thus making evasion more difficult.
ii. rapid deployment of new detection algorithms is allowed, because they do not need to be deployed at every machine in the network.

Detecting a worm outbreak is not sufficient to contain it, vulnerable computers that have not yet been infected need to be protected. Vigilante enables computers to protect themselves, but first they need to be informed about the outbreak. To do this, detectors in Vigilante generate Self-Certifying Alerts (SCAs). These are security alerts that can be verified by the computers that receive them. Using SCAs, machines cooperate to contain an outbreak, without having to trust each other.

In order to achieve this, the first step is the implementation of the containment structure, working to make it as flexible as possible so that the worms become contained in a fine grained structure. The second part is the integration of an intelligent decision-making structure into the framework, allowing the system to make rational decision according to the balance that exists between security and the operations of the computer network. Accordingly, the implementation of the Markov Chain algorithm to quantitatively evaluate the different worm containment functionalities that are available at the disposal of network administrators together with the total sum of expected positive results is proposed.

Figure 1 below was adopted from the work of Costa [1]. It depicts a Vigilante Architecture of the proposed system for worm containment. In the design, the host detects worms by

instrumenting network-facing services to analyse infection attempts. The detectors use this analysis to generate SCAs automatically and distribute them to other hosts. Before a host distributes an SCA or after it receives an SCSA from another host, it will verify the SCA by reproducing the infection process described in the SCA in a sand box. If verification is successful, the host is certain that the service is vulnerable. Alerted hosts will protect themselves by generating filters that block worm traffic before they are delivered to a vulnerable service. Each vulnerable host will run this procedure locally and installs the filter to protect itself from the worm [22].

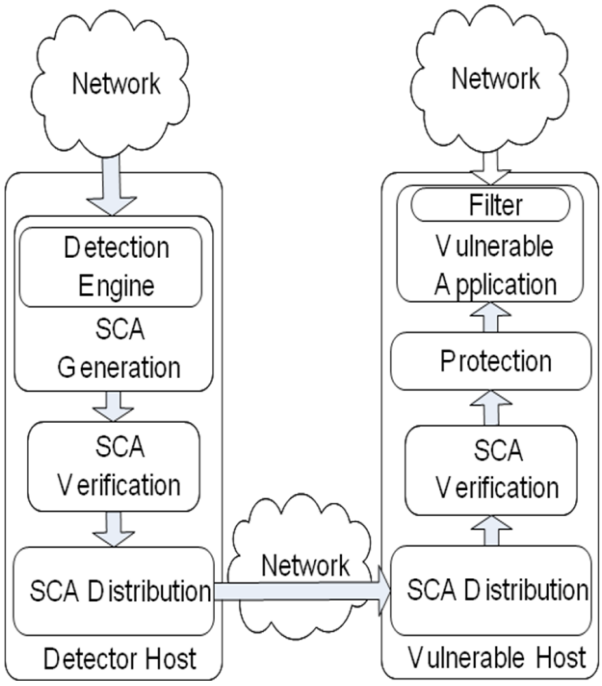

Figure 1. Vigilante Architecture [1].

The proposed system has four components which collaborate with each other to achieve the design goal as shown in Figure 1. The components are presented as follows:

i. Host based detection engine: It analyze infection attempts, there are cooperative detection without trust. Any host can be a detector, which generates Self-Certifying Alerts (SCAs), verify and broadcast SCAs.
ii. SCA generation: Detectors generate SCA when worm is detected, search log for relevant messages, compute verification information, generate tentative version of SCA and repeat until verification succeeds.
iii. SCA distribution: uses overlay of superpeers, detectors flood alerts over overlay links and hosts receive SCAs with high probability.
iv. Protection: Vulnerable hosts generate filter from SCA that block worm traffic.

Figure 2 shows the procedures for alert verification which are outlined as follows:

i. SCA verifier receives an SCA
ii. Sends the SCA to the verification manager inside the virtual machine
iii. Verification manager uses the data in the SCA to identify the vulnerable service
modifies the sequence of messages in the SCA to trigger execution of Verified
when the messages are sent to the susceptible service
iv. If Verified is executed, the verification manager signals success
v. Failure after Timeout

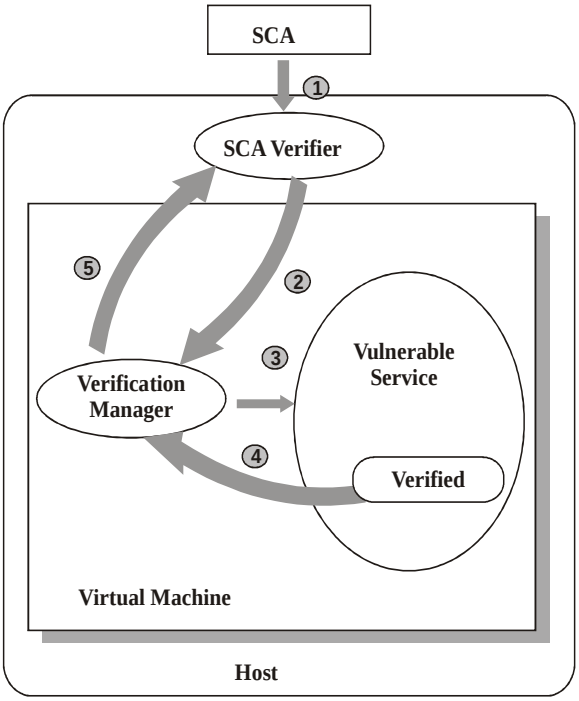

Figure 2. SCA verification component and flow of control [1].

In this research work, worm detection took place inside virtual machines (VMs). This was performed by the honey pot hosts. This makes it harder for the worm code to threaten the security of the physical system or vigilante itself.

The Algorithm was implemented on a Laboratory Local Area Network at the Information and Communication Technology Application Centre (ICTAC), Adekunle Ajasin University, Akungba Akoko with the following specifications: A Compaq D550 Pentium 4 Server running on Windows Vista Operating System with 2.4 Ghz processor speed and 2GB RAM, Fast Ethernet Network cards and the hosts were connected through a 100Mbps D-Link Ethernet switch. The Tools used for the system implementation include the Visual Basic.Net programming language.

Five computers with the above specifications were connected on the network for the experiment. One of the systems was infected with the following three commonly known worms: CodeRed, Blaster and Slammer. Tests were then carried out with regards to the processes and programs that each of the worms invokes, together with the capability of the vigilante architecture to detect when these programs are invoked and contain them.

IV. RESULTS AND DISCUSSION

An experiment was necessary to measure the time it takes to generate the Self-Certifying Alerts with the use of the algorithm as a means of detecting and generating arbitrary control over the chosen worms: Slammer, CodeRed and Blaster. In Figure 3 the SCA generation time was calculated as the time from the reception of the last message from the worms to the time when the detector generates a Self-Certifying Alert to indicate that it has noted an incoming malicious code.

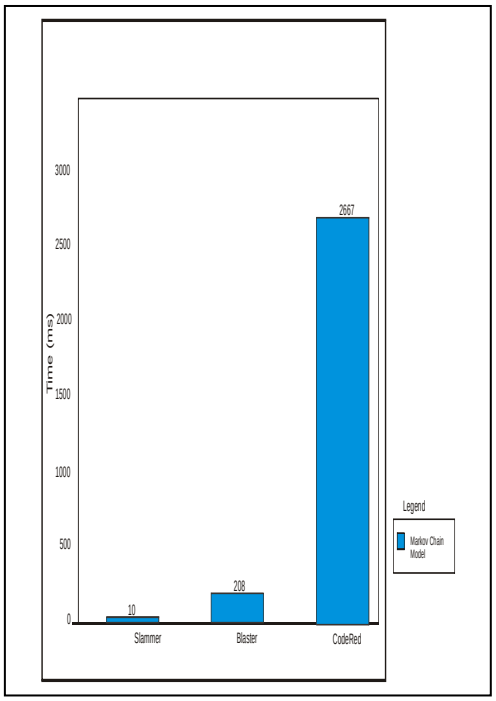

Figure 3. SCA Generation Time

Verification was done through running the worms inside a virtual machine that runs the worms and has all the code necessary for verification of the SCA. In this case, Virtual PC 2004 virtual machine was used, with the initial state of the virtual machine being stored on the disk and running the code of the worm. An average time to verify the SCA for each of the worms was developed and the result is as presented in Figure 4.

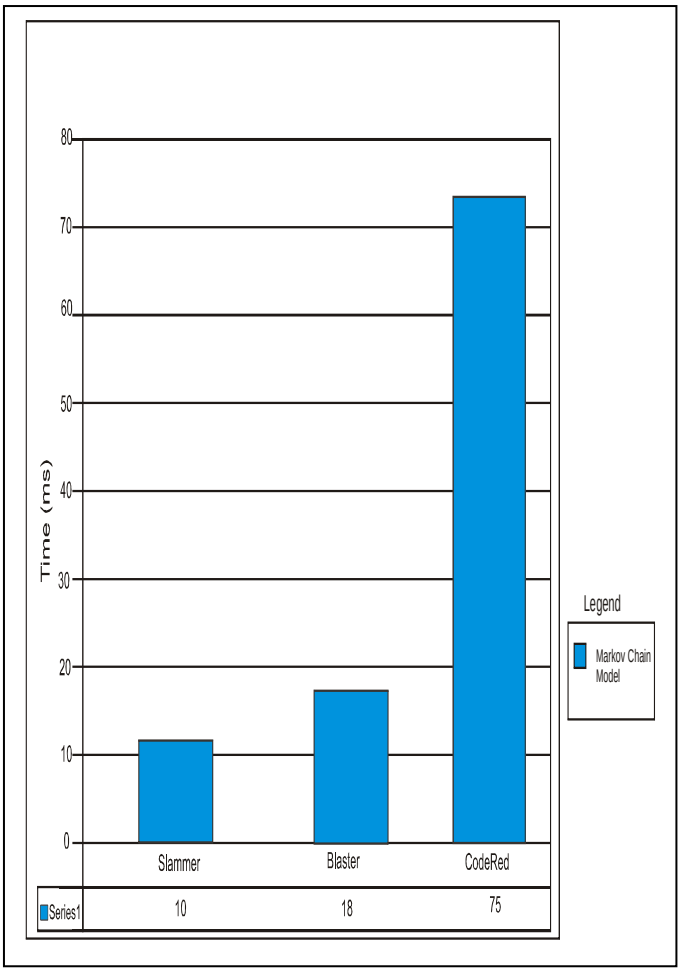

Figure 4: SCA Verification Time

The results of the experiments reveal that the Vigilante Architecture system remains effective even when there is an increase in the Self-Certifying Alert verification time and a rise in the number of nodes that are initially infected. However, the effectiveness of the system is reduced when the number of infected nodes increases. Denials of Service attacks also occur with the increase in the number of infected nodes, with their occurrence increasing with increase in the SCA verification time.

V. COMPARATIVE ANALYSIS WITH PREVIOUS WORKS

In this paper, a novel Markov Chain based Vigilante Architecture for the worm containment system is proposed. The performance of the Markov Chain Model with Dynamic Dataflow Analysis model proposed in Costa (2006) was evaluated. SCA generation time, and SCA verification time were computed for each worm (Slammer, Blaster and CodeRed) as follows: The time was calculated from the reception of the last message from the worms to the time when the detector generates self-certifying alerts to indicate that it

has an incoming malicious code. The verification time was calculated by running the worms inside a virtual machine that runs the worm and has all the codes necessary for verification of the SCA.

The average SCA generation time for each worm is shown in Figure 5. It could be observed in Figure 5 that SCA generation is faster with the proposed Markov Chain based Vigilante System for slammer worm but slower for Blaster. Similarly, SCA verification time was identical in both techniques as indicated in Figure 6. However, the SCA generation time was the same for codeRed in both techniques.

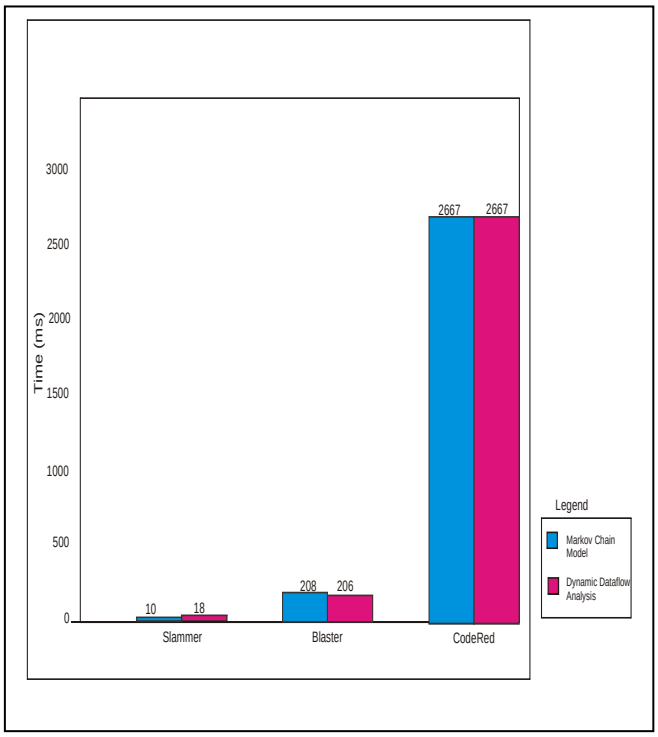

Figure 5: SCA Generation Time (Markov model Vs Dynamic dataflow Analysis)

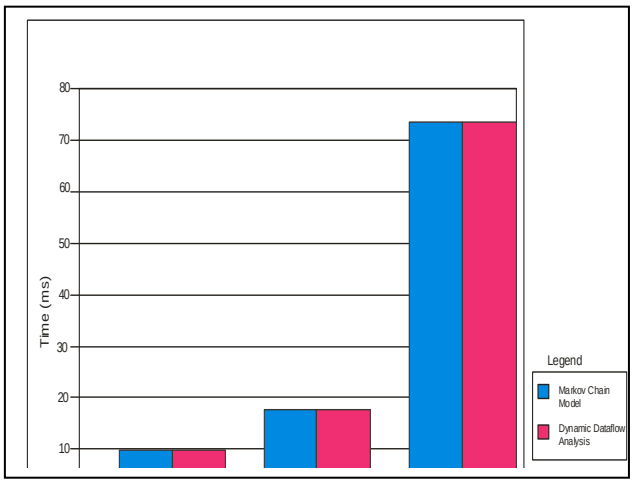

Figure 6: SCA Verification Time (Markov model Vs Dynamic dataflow Analysis)

VI. CONCLUSION

With the increase in the organizational dependence of today's society on the use of computers, it has become necessary to experiment on the deployment and containment of worms so as to come up with algorithms that will reduce their malicious effects. These worm containment algorithms have to run accurately and automatically, without necessarily causing blockage of harmless traffic in the network. In this regards, the Vigilante Architecture presents an end-to-end worm vigilante system that seeks to automate the worm containment process through analysis of incoming coded messages in the hosts and checking whether they attempt to run or access functions that they should not.

In this research, a technique for the Vigilante system approach to worm containment in a network that will generate Self-Certifying Alert (SCA) is proposed. The Self-Certifying Alert is the core idea of the Vigilante system which argues that one machine should not trust security alerts originated from other machines. Therefore, the SCA contains the necessary information to generate and verify filters, which is to be installed to prevent worm attacks. The Self-certifying Alerts present a language through which the common system vulnerabilities can be defined, allowing for the verification of detector alerts that are sent when a worm tries to run an invalid function. The verification of these SCAs presents an effective way of reducing false positives that can be created by the Vigilante Architecture.

REFERENCES

[1] M. Costa. *End-to-End Containment of Internet Worm Epidemics* Churchill College University of Cambridge, PhD Thesis, 2006.

[2] N. Weaver, V. Paxson, S. Staniford and R. Cunningham. "A taxonomy of computer worms". In *The First ACM Workshop on Rapid Malcode WORM, 2003.*

[3] D. Moore, C. Shannon, G. Voelker and S. Savage. Internet quarantine Requirements for containing self-propagating code. In *Infocom*, San Francisco, USA, 2003

[4] Nerga (2001). The advanced return-into-lib(c) exploits: Pax case study. Phrack, 2001, pp 58).

[5] S. Staniford, V. Paxson and N. Weaver. "How to own the internet in your spare time". In *USENIX Security Symposium 2002*, San Francisco, USA.

[6] V. Paxson. "A system for detecting network intruders in real time". *Computer Networks, 31(23-24 1999, pp* 2435–2463.

[7] P. Szor and P. Ferrie. "Hunting for metamorphic". In International Virus Bulletin Conference 2001.

[8] T. Holz and F. Raynal. "Detecting honeypots and other suspicious environments". In Workshop on Information Assurance and Security, 2005.

[9] J. Bethencourt, J. Franklin and M. Vernon(2005). Mapping Internet sensors with probe response attacks. In Usenix Security Symposium, 2005.

[10] D. Moore, C. Shannon and J. Brown .“Code Red: A case study on the Spread and Victims of Internet Worm.” Proc. 2nd ACM *Internet Measurement Workshop*, ACM Press, 2002.

[11] Wikipedia. Computer Networks; http://www.en.wikipedia.org, 2012

[12] B.S. Everitt. *The Cambridge Dictionary of Statistics*.

CUP. ISBN 0-521-81099-X, 2002.

[13] J.F. Shoch and J.A. Hupp. “The “Worm" Programs – Early Experience with a Distributed Computation". *Communications of the ACM, 25(3):*172-180, March 1982.

[14] M. W. Eichin and J. A. Rochlis. “With Microscope and Tweezers: An analysis of the Internet Virus". In *Proceedings of the symposium on Research in Security and Privacy*, May 1988. Oakland, CA.

[15] T.M. Chen and J. M. Robert (2004). "Worm epidemics in high-speed networks" IEEE, 2004. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1306386

[16] Symantec. "Worm.ExploreZip", 1999 http://www.symantec.com/security_response/writeup.jsp?docid=2000-121514-1418-99&tabid=1

[17] M. K. Darrell and C. E. Matthew). "Recent Worms: A Survey and Trends" ACM workshop on Rapid malcode, (2003. http://portal.acm.org/citation.cfm?id=948189

[18] S. Staniford, G.Grim and R. Jonkman. “Flash Worms: Thirty Seconds to Infect the Internet". Silion Defense - Security Information, August 2001.

[19] A. Mackie, J. Roculan, R. Russell and M.V. Velzen. “Nimda Worm Analysis”. September 2001. ARIS predictor, Attack Registry & Intelligence Service, Incident Analysis Report Version 2, Security-Focus.

[20] I. Arce and E. Levy. “An Analysis of the Slapper Worm”. *IEEE Security and Privacy*, *1(1)*, 2003, pp 82–87.

[21] M Bailey, E. Cooke, F. Jahanian, D. Watson and J Nazario. The Blaster Worm: Then and Now. *IEEE Security and Privacy*, *03(4)*, 2005, pp26–31

[22] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang and Barham. P. (2005). “Vigilante: end-to- end containment of internet worms”. In *SOSP ’05: Proceedings of the twentieth ACM symposium on Operating systems principles*, *volume 39, pp.133–147*, New York, NY, USA, ACM Press.