Dynamic Control of Worm Propagation

Ram Dantu
University of North Texas
rdantu@unt.edu

João Cangussu
University of Texas, Dallas
cangussu@utdallas.edu

Arun Yelimeli
University of North Texas
aky0001@unt.edu

Abstract

In a computer network, network security is accomplished using elements like firewalls, hosts, servers, routers, intrusion detection systems, and honey pots. These network elements need to know the nature or anomaly of the worm in priori to detect the attack. Modern day viruses like Code red, Sapphire and Nimda spread very fast. For example, Sapphire can double its size and infect more than 90% of the vulnerable hosts within 10 minutes. There fore it is impractical if not impossible for human mediated responses to these modern day fast spreading viruses. Several epidemic studies show that automatic tracking of resource usage and control is an effective method in containing the damage. In this paper we propose a state space feedback control model to detect and control the spread of these viruses by measuring the number of connections an infected host makes.

The objectives of the mechanisms are to slow down the spreading velocity of a worm by controlling (delaying) the total number of connections made by an infected host. As expected, the model showed that the sooner the infection is detected the faster the reduction of the spreading velocity. Additionally, the deployment of a controller at different levels (host and firewall) has shown to be very promising.

1. Introduction

In the past active worms have taken hours if not days to spread effectively. This gives sufficient time for humans to recognize the threat and limit the potential damage. This is not the case anymore. Modern viruses spread very quickly. Damage caused by modern computer viruses (example - Code red, sapphire and Nimda) is greatly enhanced by the rate at which they spread. Most of these viruses have an exponential spreading pattern. Future worms will exploit vulnerabilities in software systems that are not known prior to the attack. Neither the worm nor the vulnerabilities they exploit will be known before the attack and thus we cannot prevent the spread of these viruses by software patches or antiviral signatures [1].

Some of the objectives of this paper are to design a State space control model to: (1) detect worm based threats; (2) dynamically quarantine infections to localized sectors to prevent propagation of infection and develop technologies to automatically and dynamically quarantine these fast spreading worms to a peak infection portion of 1% of vulnerable machines that would otherwise infect approximately 100% of vulnerable machines.

The model developed is based on the analysis of some popular worms that spread very fast caused havoc in the recent history. Below is a brief description of these worms, their spreading pattern and some of the techniques these worm authors' use to increase the spread rate.

Code Red I: On July 12, 2001, this worm began to exploit the buffer-overflow vulnerability in Microsoft's IIS web servers. Upon infecting a machine, the worm checks to see if the date (as kept by the system clock) is between the first and the nineteenth of the month. If so, the worm generates a random list of IP addresses and probes each machine on the list in an attempt to infect as many computers as possible. Once Code-Red infected a host, it spread by launching 99 threads, which generated random IP addresses, and then tried to compromise those IP addresses using the same vulnerability. In some cases a hundredth thread defaced the web server [3].

Code Red II: This worm also used the same Microsoft's IIS web server's buffer overflow vulnerability as Code Red I. This worm had single stage scanning technique and used a localized scanning strategy, which made it successful in infecting addresses close to it. It chose a random IP address from within the class B address space (/16 network) of the infected machine with probability of 3/8 and it chose from its own class A (/8 network) with probability ½ and finally a random address from the whole internet with a probability 1/8. This

strategy was very successful, because there was more probability of finding vulnerable machines within a network, once it has passed through the external firewall. [3]

Nimda: this worm had a very ferocious spreading strategy. It used multiple ways to spread through the network. This worm is believed to have used the following five different strategies to spread

1. "By infecting Web servers from infected client machines via active probing for Microsoft IIS vulnerability.
2. By bulk emailing of itself as an attachment based on email addresses determined from the infected machine.
3. By copying itself across open network shares.
4. By adding exploit code to web pages on compromised servers in order to infect clients which browse the page.
5. By scanning for the backdoors left behind by Code Red II and also the "sadmind" worm" [3]

Sapphire: sapphire worm was one of the fastest worms in history. It was so fast, it doubled in size every 8.5 seconds and infected more than 90% of the vulnerable hosts within 10 minutes. This worm also used a buffer overflow vulnerability of Microsoft's SQL server. It infected more than 75000 hosts and caused extreme consequences such as cancelled airline flights, interference with elections and ATM failures [4].

One of the most significant features of Sapphire was its speed of spread. The worm achieved full scanning rate (over 55 million scans per second) in approximately three minutes. It slowed down after initial exponential rate due to bandwidth constraints of the network but most vulnerable machines were infected within 10 minutes of the worm's release [4].

Sapphire used random scanning strategy to spread though the network. Worms using random scanning strategy have exponential pattern at the beginning of the spread and slow down later as they try to infect the same machine again and again.

Response to Sapphire's spread

Most accurate data was obtained from the University of Wisconsin Advanced Internet Lab, where all packets into an otherwise unused network (a "tarpit" network) are logged.

Many sites began filtering all the UDP packets with a destination port of 1434. Though filtering reduced the bandwidth consumed by the infected hosts, it did nothing to limit the spread of the worm.

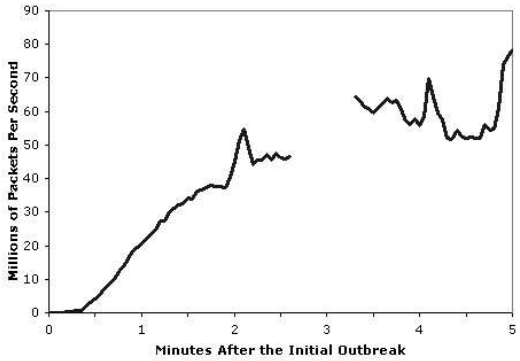

Figure 1. Aggregate Scans/Second in the first 5 minutes based on incoming connections to the WAIL Tarpit.

The response was so slow that by the time filtering was implemented; the worm had infected almost all the susceptible hosts [4]. Figure 1 & 2 shows the scanning rate of this virus in the first five minutes and first twelve hours after the launch of the worm.

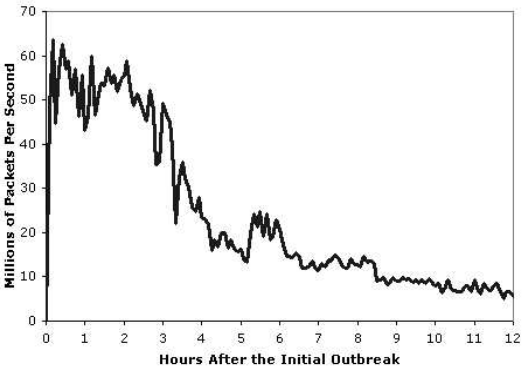

Figure 2. Aggregate Scans/Second in the 12 hours after Initial outbreak [4].

Distributed Denial of service attacks

This can be described as attempt by an attacker to prevent legitimate users from using resources. An attacker usually floods the network and steals the bandwidth available to the user.

Some well-known examples of denial of service attacks include:

- "Attempts to "flood" a network, thereby preventing legitimate network traffic.

- Attempts to disrupt connections between two machines, thereby preventing access to a service.
- Attempts to prevent a particular individual from accessing a service.
- Attempts to disrupt service to a specific system or person" [2].

V. Paxson *et al* describes some of the techniques, which could be used by the worm authors to enhance the spread rate of worms.

"Better" worms - theory

There are several ways a worm spreads through the network. Some of the common techniques employed by the worms are discovering more widespread security holes and increasing the scanning rate. Apart from these some of the strategies that a worm author could adopt are:

(i) Hit-list scanning.
(ii) Permutation scanning.
(iii) Topologically aware worms.
(iv) Internet scale hit lists.

The ultimate goal of all these strategies is to spread the worm as fast as possible.

(i) Hit-list scanning: though most of the worms propagate exponentially, it is the initial take off time that is difficult. It takes more time to infect the first 1000 machines. The strategy to overcome this problem is called Hit list scanning. In this strategy the worm author collects a list of 10,000 to 50,000 potentially vulnerable machines.

Some of the ways the worm author collects these list of vulnerable machines are:

- Stealthy scans – it can be obtained by scanning the entire Internet.
- Distributed scanning – Using this strategy the attacker can scan a few dozen to few thousand already-compromised "zombies".
- DNS searches – A list of domain names can be obtained and then their IP addresses from domain names.
- Spiders – Use of web-crawling techniques similar to search engines to get a list of most interconnected web sites.
- Public surveys - there are surveys to a get a list of potential targets.
- Just listen – Some applications like peer-to-peer networks advertise their servers, also previously effective worms broadcast the vulnerable machines.

(ii) Permutation scanning: one of the main problems faced by random scanning was that many infected machines were scanned many times wasting time. This problem was overcome in permutation scanning where all worms share a common pseudo random permutation of the IP address space.

(iii) Topological scanning: This is an alternative approach to obtain a set of vulnerable IP addresses. This method uses information contained in the victim's machine.

(iv) Flash worms: This is an alternative to hit-list scanning discussed. Using an OC12 connection all the web servers can be scanned within 2 hours. The list is divided into n blocks. After infecting a host, the worm hands over the list to a child worm, which goes on and infects that particular block. Thus parallel spreading can be achieved and hence faster worms [4].

Existing Defenses

There are a variety of network components in the market today that protect machines from different kinds of worms. Network perimeter is primary concern for security managers. Security *managers* have focused on multiple security components to keep their networks safe. Examples are: firewalls, intrusion detection systems, and honey pots.

Intrusion Detection Systems: intrusion detection systems are like burglar alarms. There are two types of intrusion detection systems, one is network and another is host. Network intrusion detection systems examine the network traffic and host intrusion systems detect outsider infiltration as well as unauthorized access by users who are trusted insiders. Intrusions are characterized into network traffic patterns that are suspicious and these are called signatures. These signatures are compared against the network traffic patterns and deviation generates security alerts. But these alerts can be false alarms. Due to nature of the signatures, these systems can be as accurate as the signatures themselves. Moreover, these systems are reactive and cannot prevent the attacks [5].

Firewalls: Currently there are several kinds of firewalls deployed in the perimeter of the networks. These are: static packet filters, dynamic packet filters,

circuit level packet filters, proxy gateways, and stateful inspection firewalls. Proxy firewalls often only have the packet filter rules applied or are used to protect just one server, such as external web server. Hence most of these firewall rules are static and cannot respond to dynamic changes [5].

Firewall Routers: Routers with firewalls are great for cost containment. However, they add complexity and overhead to the router's function. Many security experts are concerned in having all security in one box [5].

Honey Pots: Honey pots lure attackers by presenting a more visible and apparently vulnerable resource than the enterprise network itself. These are also useful for forensics. But these can be vulnerable themselves because they attract attackers special attention. Also if they are incorrectly configured, they make network more vulnerable [5].

Thus firewalls, routers, Intrusion detection systems, and honey pots can be very useful as elements for network defense but they can not protect the network by themselves. But by careful integration and engineering of these devices, security level can be increased [5].

2. Methodology

There is a need to control the spread of above mentioned, fast spreading viruses automatically. They spread very fast for human initiated control. Some of the automatic approaches like quarantining the systems and shutting them down reduce the performance of the network. False positives are one more area of concern [1].

Williamson [1] describes a novel approach to this problem. This situation can be improved a lot by using "benign" responses, those that slow but do not stop the virus. The main idea is to delay the virus by so long as to earn time for human mediated responses [1]. Feedback control strategy is desirable in such systems because well-established techniques exist to handle and control such a system [15].

This technique is based on the fact that an infected machines tries to make connections at a faster rate than the machine that is not infected. The idea is to implement a filter, which restricts the rate at which a computer makes connection to other machines. The delay introduced by such an approach for normal traffic is very low (0.5 –1 Hz). This rate can severely restrict the spread of high-speed worm spreading at rates of at least 200 Hz [1].

As a first step towards our design, we apply feedback control to the first level of hierarchy (node). We will then expand the model to further levels as shown by the system architecture in the next section. By careful integration, engineering, measurement and control of these devices, network security level can be increased [5][6]. We have implemented classical feedback control theory model for network-level security management.

Advantages of feedback control are: (i) System (e.g., group of security components in a network) output can be made to follow the specified function in an automatic fashion; (ii) System performance is less sensitive to variations of parameter values; and (iii) Use of feedback makes it easier to achieve the desired transient and steady-state response.

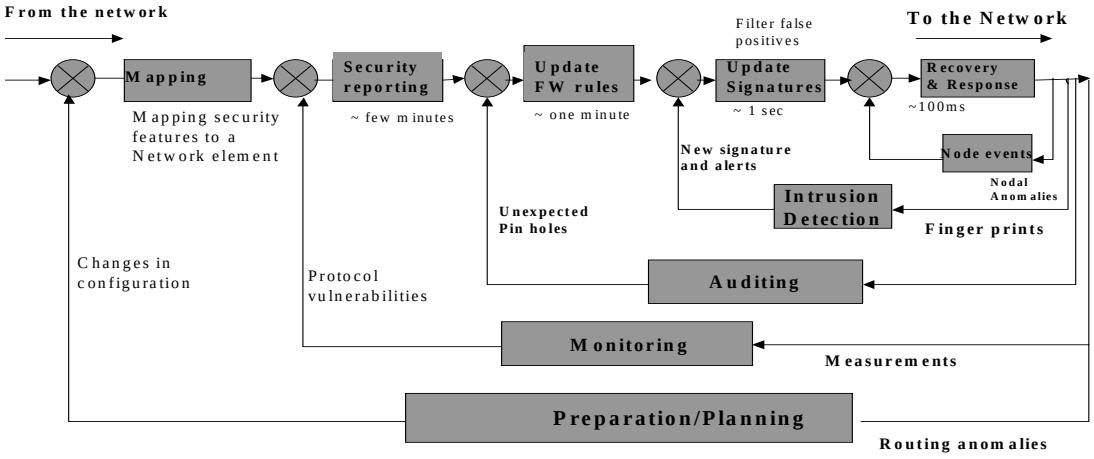

Figure 3. Controller architecture for end-to-end security engineering

System architecture

It is assumed a secured network consists of firewalls, sensors, analyzers, honey pots, and various scanners and probes. These components are either separate elements or collocated with hosts, servers, routers and gateways.

In this architecture, a (centralized or distributed) controller is responsible for collection and monitoring of all the events in the network. This controller is knowledgeable about the network topology, firewall configurations, security policies, intrusion detections and individual events in the network elements. This controller is logical function and can be deployed anywhere in the network.

As shown in Figure 3, the controller communicates with clients located in different network elements. Clients are responsible for detection and collection of the events in the node and communicate to the controller. Subsequently, controller will run through the algorithms, rules, policies and mathematical formulas for next course of action. These actions are communicated to the clients.

As described in Figure 3, the architecture evolves from a concept of closed loop control. Changes regarding the security behavior are captured and mixed with the incoming network signals. This piece of information is used to formulate the next course of action. The final result is outcome from multiple loops and integration of multiple actions. The response times within each loop are indicated in Figure 3 (we call them defender-loops)**.** Response time varies from few milliseconds to several tens of minutes. For example, nodal events like buffer overflows, performance degradation can be detected in matter of milliseconds. On the other hand, it may take several seconds to detect failed logins, changes to system privileges and improper file access.

We split the number of connection requests into three categories.

- Connections to machines that are considered safe (a queue is maintained for safe list connections)
- Connections that are delayed (All the connection not present in the safe list but can go through)
- Connections that are dropped for other reasons

Next, a state model for the diagram shown above is presented where control is achieved by varying model parameters.

State Model

Here it is assumed that, as the number of requests increases, a portion of them will be sent to a delay queue to be served later. Parameters related to the size of the delay queue and the number of dropped connections are used to control the total number of connections resulting in a slow down of spreading worm.

Sapphire worm spreading is taken as an *example* to show the applicability of our approach. The goal of the example is to slow down the spreading velocity of a worm by controlling the total number of connections (C(t)) detected by the host. A model capturing the behavior of the system, i.e., how the number of total connections is changing is needed to achieve this goal. We assume here that as the number of request increases, a portion of them will be sent to a delay queue to be served later. Parameters related to the size of the delayed queue and the number of dropped connections are used to control the total number of connections resulting in a slow down of a spreading worm [6][7].

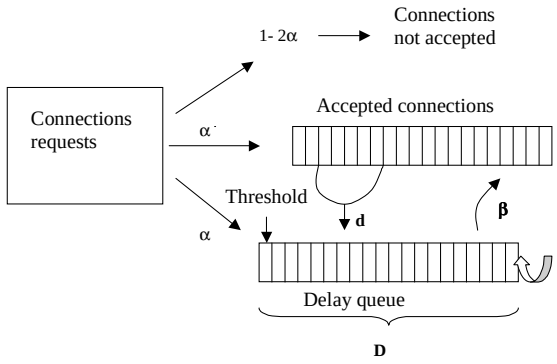

Figure 4. Model for connections accepted, delayed and rejected for parametric control.

The rate of change of the number of connections (dc/dt) is proportional to the number of dropped connections (-dc, where d is the specified drop rate) plus the number of connections removed from the delayed queue (βD, where β is delay parameter and D is the number of delayed connections on the queue) and the new successful connections (αu, where α is the percentage of not delayed connections). This results in

$$\dot{c} = -dc + \beta D + \alpha u \qquad (1)$$

Differentiating Eqn. 1 we obtain

$$\overset{\circ\circ}{c} = -d[\overset{\circ}{c}] \quad (2)$$

Substituting for $\overset{\circ}{c}$ we obtain

$$\overset{\circ\circ}{c} = -d^2 c + -d\beta D + d\alpha u \quad (3)$$

The rate of change in the size of the delayed queue (dD/dt) is proportional to the new incoming successful connections send to the queue (αu) minus the connections removed from the queue -βd. This results in

$$\overset{\circ}{D} = -\beta D + \alpha u \quad (4)$$

Where D is the size of the delay queue; d is the drop rate; β is the delay parameter; u is the total connections arriving; and α is the success rate

Combining equations for dC/dt and dD/dt in state variable format leads to

$$\begin{bmatrix} \overset{\circ}{c} \\ \overset{\circ\circ}{c} \\ \overset{\circ}{D} \end{bmatrix} = \begin{bmatrix} -d & 0 & \beta \\ d^2 & 0 & -d\beta \\ 0 & 0 & -\beta \end{bmatrix} \begin{bmatrix} c \\ \overset{\circ}{c} \\ D \end{bmatrix} + \begin{bmatrix} \alpha \\ -d\alpha \\ \alpha \end{bmatrix} [u] \quad (5)$$

$$\begin{bmatrix} \overset{\circ}{C} \\ C \\ \overset{\circ\circ}{C} \\ D \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ d^2 & 0 & -d\beta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c \\ \overset{\circ}{c} \\ D \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -d\alpha \\ 0 \end{bmatrix} [u] \quad (6)$$

The output vector of Eq. (6) presents the results for the computation of the total number of connections C(t) as shown in Figure 6a, the rate of change of the first derivative of C(t), that is, the acceleration (Figure 6b), and the number of connections on the delayed queue (Figure 6c). The input in Eq. (6) represents the number of requested new connections per time unit. In the case of a normal traffic the average number of requests can be considered constant over a period of time. However, an S-shape form is expected for the spread function of a worm and dU/dt=U(1-U) is used to generate the input. The solid line in Figure (6) represents the behavior of the system with drop rate and delay parameters under "regular" conditions. A question is of keen interest here: "How to use the output of the model to detect a spreading worm?" Once an

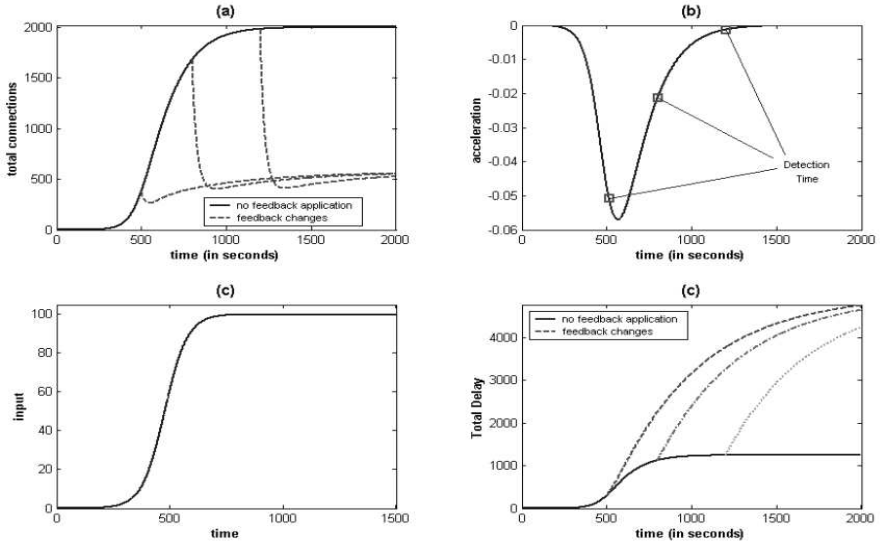

Figure 6: Behavior of the state model for the control of the number of connections on the presence of a worm spreading according to an S-shape function. (a) Shows the total number of connections with and without feedback; (b) shows the acceleration and the detection times; (c) shows the number of connections on the delayed queue; and (d) shows the results of the application of the feedback loop approach at the host and at the firewall level.

infection is identified, feedback control can be used to reduce the velocity of the spreading function.

The dashed lines in Figure 6 represents the results for different detection times of the application of feedback control for the scenario described above.

The acceleration value of the number of connections is used here as a detection mechanism. That is, when the acceleration reaches a certain threshold the drop rate and the delayed parameters on the model in Eq. (6) are adjusted to slow down the spreading of the worm. Figure 6a shows the results of this detection mechanism for three distinct values for the threshold. As expected, the sooner the infection is detected the faster the reduction of the spreading velocity.

Now, consider a scenario where hosts are connected to the firewall and a controller is available at the hosts and at the firewall. The control can be done at the host level, at the firewall level, or at both levels. Figure 6 shows the results of applying or firewall) converges to the same results though a larger overshoot is observed at the firewall level. Regarding detection time, the acceleration at the firewall level increases faster than at the host level and consequently an earlier detection time is expected at the firewall level. As observed from Figure 6, a double feedback loop has the advantage of the early detection time at the firewall level and a more effective result in slowing down the infection.

As shown in the graph the system was tested to check the behavior by applying control at different stages on spreading.

4. Design & Implementation

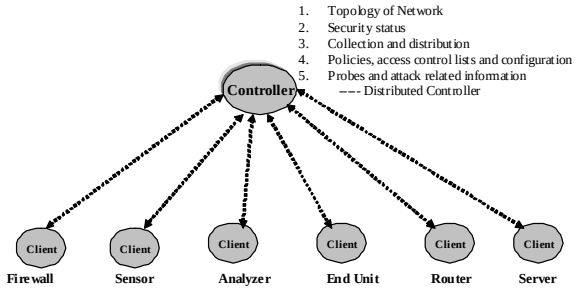

Figure 9: Communication between various elements for meeting end-to-end security requirements

It is assumed a secured network consists of firewalls, sensors, analyzers, honey pots, and various scanners and probes. These components are either separate elements or collocated with hosts, servers, routers and gateways.

In this architecture, a (centralized or distributed) controller is responsible for collection and monitoring of all the events in the network. This controller is knowledgeable about the network topology, firewall configurations, security policies, intrusion detections and individual events in the network elements. This controller is logical function and can be deployed anywhere in the network.

As shown in Figure 8, the controller communicates with clients located in different network elements. Clients are responsible for detection and collection of the events in the node and communicate to the controller. Subsequently, controller will run through the algorithms, rules, policies and mathematical formulas for next course of action. These actions are communicated to the clients.

5. Conclusions

In this paper, we have proposed a security controller based upon the feedback control theory. The presented state model has shown to be completely controllable and observable in addition of being stable. Several measurements, signatures, topological changes, finger prints are continuously extracted from the network. These measurements are fed back to the feedback control loop and compared with the existing conditions and an error is generated. This error signal is used to calculate the transfer function of the feedback loop and update the security components in the network. For example, new policy rules are added, deleted and new honey pots may be created.

To satisfy the feedback control loop, a specification and requirements for expected output need to be specified. Also, certain measurements, benchmarks, and metrics are specified for satisfying these requirements. Likewise, certain buffer size, CPU utilizations are also specified. Further work involves specification of requirements**,** and benchmarks, and deriving transfer functions for each module in the feedback loop.

6. References

[1] Williamson, M.M, "Throttling Viruses: Restricting propagation to defeat malicious mobile code", *Computer Security Applications Conference, 2002. Proceedings*. 18th Annual, 9-13 Dec. 2002 Page(s): 61 –68

[2] Lau, F.; Rubin, S.H.; Smith, M.H.; Trajkovic, L.;, "Distributed denial of service attacks", *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, Volume: 3 , 8-11 Oct. 2000, Page(s): 2275 -2280 vol.3

[3] S. Staniford, V. Paxson, and N. Weaver. "How to own Internet in your spare time", *In Proceedings of the USENIX Security Symposium*, pages 149--167, August 2002.

[4] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, Nicholas Weaver, "The spread of the sapphire worm", http://cs.berkely.edu/~nweaver/sapphire

[5] R.V. Dantu, "An Architecture of Security Engineering", ACSA *Workshop on Application of Engineering Principles for Security System Design*, November, 2002.

[6] R.V. Dantu "Feedback control for network security engineering", *18th Annual ACSCA Conference on practical solutions to security engineering*, December 2002.

[7] João W. Cangussu, Ray A. DeCarlo, and Aditya P. Mathur, "A Formal Model of the Software Test Process", *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 782-796, August, 2002.

[8] Somayaji, A., and Forrest, S., "Automated response using system-call-delays", *Proceedings of the 9th USENIX Security Symposium*, 2000.

[9] João W. Cangussu, Ray A. DeCarlo, and Aditya P. Mathur, "Using Sensitivity Analysis to Validate a State Variable Model of the Software Test Process", *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 782-796, May, 2003.

[10] Graham C. Goodwin, Stefan F. Graebe and Mario S. Salgado, *"Control System Design"*, Prentice Hall, 2001.

[11] João W. Cangussu, Ray A. DeCarlo, and Aditya P. Mathur, *"A Sate Model for the Software Test Process with Automated Parameter Identification"*, Proceeding of the 2001 IEEE Systems, Man, and Cybernetics. Tucson, Arizona, pp. 706-711.

[12] João W. Cangussu, Ray A. DeCarlo, and Aditya P. Mathur, "Feedback Control of the Software Test Process Through Measurements of Software Reliability", *Proceedings of the 12th IEEE International Symposium on Software Reliability Engineering*, pp. 232-241, Hong Kong.

[13] David Moore, Colleen Shannon, Geoffrey M. Voelker, Stefan Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code", *INFOCOM*, 2003.

[14] N. Gandhi, D.M. Tilbury, Y. Diao, J. Hellerstein and S. Parekh, "MIMO Control of an Apache Web Server: Modeling and Controller Design", *Proceedings of the American Control Conference*, Anchorage, AK May 8-10, 2002.