REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
15-01-2009	FINAL PERFORMANCE REPORT	Sep 2005 - Sep 2008

4. TITLE AND SUBTITLE
Emulation of the Active Immune Response in a Computer Network

5a. CONTRACT NUMBER
FA9550-05-1-0361

5b. GRANT NUMBER
40872

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Skormin, Victor A.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Center for Advanced Information Technologies
Binghamton University
Binghamton, NY 13902

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-Final-3

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
Suite 325, Room 3112
875 N. Randolph Street
Arlington, VA 22203-1768

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0493

12. DISTRIBUTION/AVAILABILITY STATEMENT
Available to general public – A

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Active immune response (AIR) offer principles that could be successfully implemented in defense mechanisms of computer networks of the nearest future: decentralized detection/mitigation, portable specialized defense agents, continuous status assessment, feedback mechanism assuring a rational parity between attacking and defense agents, et. A mathematical model of AIR has been developed and validated by simulation. On its basis, a computer network defense mechanism utilizing the above features, capable of deployment of specialized anti-worm entities in a computer network, is proposed. It leads to the development of a fully automatic computer network defense system. The feasibility and implementation aspects of the particular components of this system are addressed. A system call-based approach resulting in the improved IDS, and an approach to continuous status assessment of a computer networks by selective scanning are developed.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (Include area code)
U	U	U	UU		

Standard Form 298 (Rev. 8/98)
18

20120918138

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

Modern immunology provides a detailed but rather qualitative description of the active response of the immune system to any entity invading the biological organism and recognized as an antigen by the immune cells. In a similar way that a malicious computer program is constructed from the same instructions as legitimate software but sequenced in a fashion that makes it malicious, an alien proteins are constructed from the same building blocks (amino acids) as the cells of the host but differently sequenced. In many ways, the effects of an antigen on a biological organism are similar to those caused by some information attacks on computer networks. This is the major reason for considering the active response of the immune system, honed to perfection by million-year evolution, as an ideal mechanism for protecting computer networks from information attacks. It is said that the immune response is genetically optimized for its specific environment; although this environment is different from a computer network, this difference narrows with every new advancement in computing technology.

The specific immune response is the main mechanism enabling the immune system to destroy cells of the intruding antigen. This is accomplished by multiplying, on demand, fighter cells that are uniquely equipped for counteracting this particular antigen by carrying the genetic sample of the intruder. The immune system is prepared to counteract practically any antigen as it contains cells that specialize in at least 10^{15} various genotypes. However, the actual ability of the immune system to destroy an intruder depends on its ability to detect and identify the intruder, generate specialized fighter cells at the necessary rate maintaining the necessary balance between the concentration of the intruder and immune cells, i.e. to actively respond to the intruder.

Consequently, active response of the immune system includes several stages. It starts from the intrusion of the antigen cells in the biological organism. Intruding cells quickly proliferate and their concentration exponentially increases. As the result, the probability of a physical contact of an antigen cell with a specialized immune cell capable of recognizing it as an antigen increases. The initial concentration of specialized cells depends on the previous exposures of the organism to this antigen (i.e. acquired immunity). The detection of the antigen triggers the process of exponential proliferation of immune cell-fighters specialized to destroy the antigen cells. Multiplying antigen cells and multiplying immune fighter cells compete for limited resources of the biological organism. It could be seen that the outcome of this process (recovery, chronic infection or lethality) greatly depends on the time period between the moment of infection and the moment of detection of the antigen. When the proliferation of the antigen cells goes too long before detection, its cells consume a greater share of the resources of the host thus preventing the fighter cells from sufficiently multiplying, leading to lethality. A high initial concentration of specialized fighter cells (after the organism has been immunized for a particular infection) facilitates the early detection of the antigen and prevents it from overwhelming the immune defenses. After the antigen has been defeated, the residual concentration of specialized fighter cells slowly decreases providing high immunity to similar infection. It could be seen that at certain conditions, parity between proliferating antigen and fighter cell could be achieved leading to chronic infection. Figure 1 below illustrates stages of the active immune response.

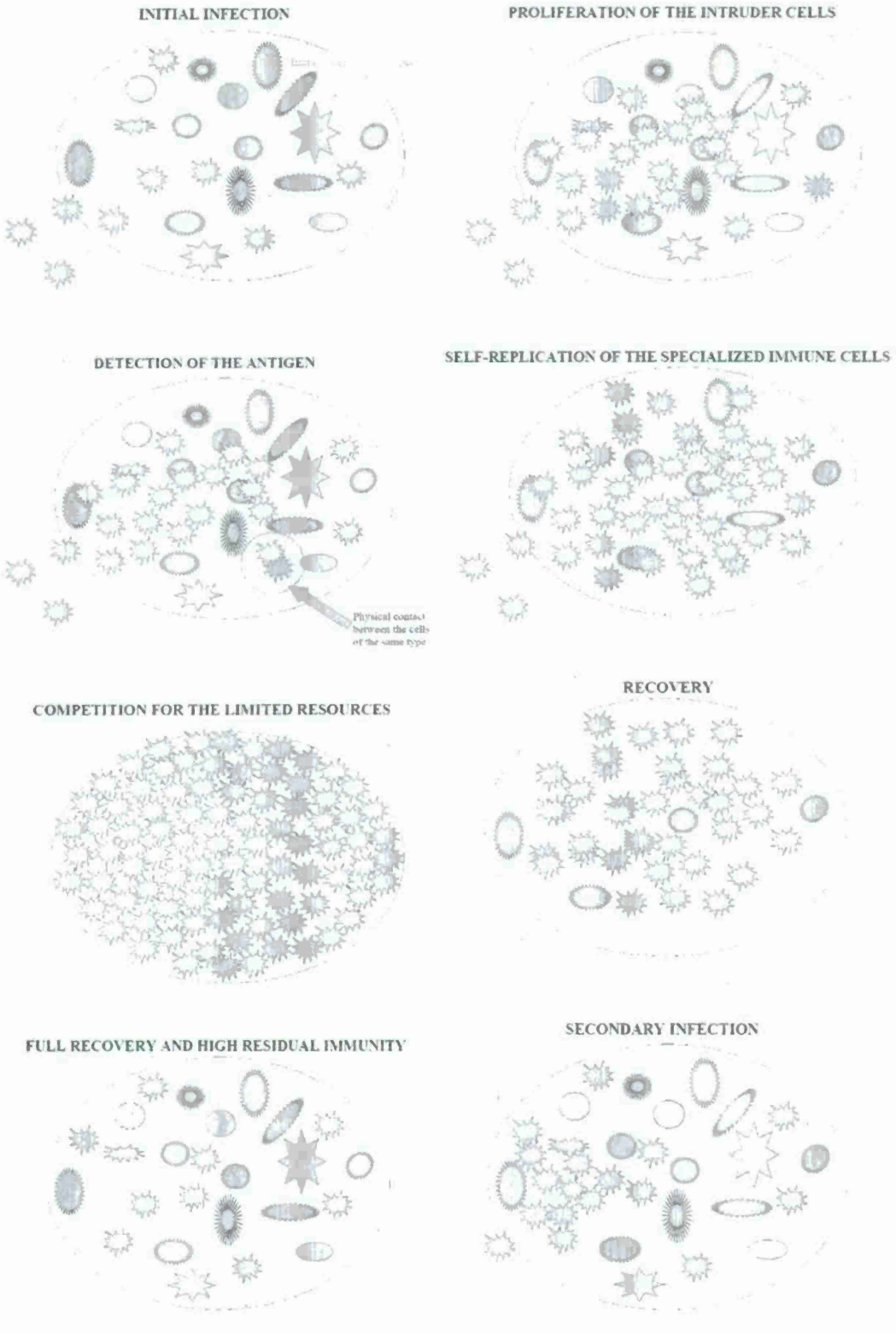


Figure 1: Stages of Active Immune Response

The mechanism of active response of the immune system could be described in terms of a negative-feedback closed-loop circuit and nonlinear differential equations could be established for its particular stages. This would result in a mathematical model of the immune response describing the complex interaction of three major factors, concentration of the antigen cells, concentration of the specialized immune fighter cells, and the available resources of the organism. Such a model would provide a means for the understanding in quantitative terms and numerical simulation of the active immune response. Subjected to analytical techniques offered by modern control theory, this model would enable us to establish the conditions for three possible outcomes of such interaction, full recovery, chronic infection, and lethality, and to formulate a control law assuring the full recovery outcome.

It is understood that the proposed model will be only as accurate as its parameters that in the case of the biological immune system could be only roughly guessed at. Therefore, the model cannot be effectively used for the prediction of the outcome of a disease caused by antigen. It is equally meaningless to utilize the developed model for the modification of the immune response as it is well beyond our abilities. However, the importance of the model of the immune response for the computer network applications shall not be underestimated because of the following reasons:

- this model would describe the principle of operation of a system providing very successful defenses against information attacks;
- in the case of a computer network, parameters of such a model could be accurately estimated that makes the model accurate and dependable;
- such a model could be instrumental for the analysis and design of defense mechanisms intended for computer networks.

Therefore, at the second stage of this research it is proposed to utilize the developed model for the description of the infection and recovery of computer networks subjected to information attacks. One can realize that every stage of the active immune response could be observed in our experience with recent computer epidemics. With the appropriate choice of parameters, the model of immune response presents a successful simulation tool for the vulnerability analysis and prediction of consequences of information attacks on computer networks. Moreover, the control law synthesized using the methodology of advanced control systems to assure a full recovery outcome for the immune system could be reformulated in terms of the characteristics of computer networks and interpreted as a set of instructions to a network manager.

This research is aimed at

1. Establishing a mathematical model of the immune response utilizing recent findings of immunology and implementing it in a simulation environment;

2. Parameter selection of the model in order to assure "credible" simulation of the interaction between the antigen and the immune cells described by the three major groups of model variables: concentration of the antigen, concentration of the specialized immune cells, and the available resources of the organism.

3. Applying the methodology of modern control theory for the establishing the principles of operation of a closed-loop control system assuring the survivability of a computer network subjected to an information attack perpetrated by self-replicating malicious software

4. Addressing the feasibility issues pertaining to the development of a computer network defenses implementing the active immune-like response to self-replicating malicious software

2.0 IMMUNOLOGY-INSPIRED METHODS IN COMPUTER SECURITY

Our ever-growing dependence on computer networks is accompanied by ever-growing concerns about the networks' vulnerability to information attacks and the dependability of the existing network security systems. Major threats, well recognized by government, private institutions and individual users, are stemming primarily from self-replicating malicious software. Modern worms and viruses propagate through the Internet much faster and cause more damage than their predecessors. In 2001 the Code Red worm propagated faster than the Melissa virus in 1999 and much faster than Morris' worm in 1988. In the case of the Code Red worm, only several days passed from the moment of its first detection to a wide spread propagation of malicious activity. Several months later, the Nimda worm caused severe damage within one hour of the detection of infection. The Slammer worm caused harm in only a few minutes. Since Code Red, the development of complex infection strategies has progressed to such a level that in January 2003, the W32.SQLExp worm propagated so fast that human intervention could not prevent its spread. A governmental reaction to the threat of virus activity and the destruction that could ensue was to create and fund organizations such as the Coordination Center of Reaction to Computer incidents (CERT/CC) in 1988 and the Department of National Cyber-Security in 2003.

Up until now effective counter-measures to worm attacks have consisted of revealing the infected hosts as quickly as possible in order to minimize damage and search for the "holes" in security systems. However, there are many factors which decrease the efficiency of counter-measurers. Every year about four thousand new "holes" in security systems are revealed. Presently, more than 200 million computers are connected to the Internet and their numbers are growing rapidly. Every moment millions of vulnerable computers are interconnected through the Internet. Sophisticated attacks can provide resources to adversaries allowing them to utilize the vulnerable computers to aide in carrying out future mass attacks. Many attacks are performed in a completely automatic fashion and are distributed at the speed of light throughout the Internet disregarding geographical and national borders. Technologies utilized by developers of malicious software are becoming more and more complex and in some cases are completely concealed from detection which has the effect of increasing the time necessary for the detection and analysis of the attack. There is a growing dependency on the Internet among its users and many rely heavily on it for business transactions as well as many other important functions. Even a relatively short-term interruption of Internet services can cause significant economic damage and may subject major governmental and private services to threat. Due to the combination of these factors even with the fastest reaction, it is quite reasonable to expect that a major cyber attack would cause significant economic losses and a decline in overall workplace productivity.

Computer network security has been a major area of concentration of research efforts. While "traditional" research utilizing methods of computer science, information science, cryptology, communication, etc. is still prevalent, a novel direction in computer security was established over two decades ago. It was realized that by its very nature, a typical attach on computer network has many commonalities with the attack of an alien protein on a biological organism. The similarity between the biological immune system and a computer network security system could be demonstrated by just a few common features tabulated below [1], [2]. This explains the attention to the development of network virus propagation models reflecting methodologies and terminology acquired from biological immune systems. It is becoming increasingly common to

find solutions, inspired by epidemiology and immunology, applied to various intrusion detection/mitigation systems for computer/networks successfully dealing with malicious software.

Table 1. Similarity Between Biological Systems and Computer Networks

Biological Systems	Computer Networks
High complexity, high connectivity, extensive interaction between components, numerous entry points	High complexity, high connectivity, extensive interaction between components, numerous entry points
Vulnerability to alien microorganisms that can quickly contaminate the system resulting in its performance degradation and collapse	Vulnerability to malicious codes introduced in the system result in unauthorized access to information and services and denial of service
Alien microorganisms as well as cells of a biological system are composed of the same building blocks, a small number of basic amino acids	Malicious and legitimate software are composed of the same building blocks - basic macro commands
The difference between alien microorganisms and the healthy cells of a biological system is in the (gene) sequencing of their building blocks	The difference between malicious codes and the legitimate software of a computer network is in the sequencing of their building blocks
Invading alien microorganisms proliferate within the host consuming its vital resources	Information attacks are perpetrated by self-replicating malicious software that consumes critical resources of the computer network in order to proliferate
Immune defenses compose a multitude of semi-autonomous specialized, cooperating and communicating agents – immune cells	A multitude of semi-autonomous specialized, cooperating and communicating agents – is the format of modern software systems

In 1991 Kephart pioneered the application of epidemiological models for the mathematical description of the complex dynamic phenomenon of propagation of self-replicating software such as simple file viruses [3]. File viruses distribution in networks was formalized in terms of probability laws [4, 5] for homogeneous, localized and random replication patterns. He should be credited for the introduction of the very concept of immune system for computers in [6] and its further development in [7] and [8], [9]. Worms have received true recognition after the attack of the Code Red worm in July, 2001. Consequently, the first propagation case study was presented in [10], where authors utilized the collected data for the analysis of the infection and disinfection rates. More fundamental analysis of the worm propagation dynamics was performed for SQL Slammer worm in [11].

The deterministic and stochastic nonlinear mathematical models developed by Kephart [3-7], alongside with the development of the theory of mathematical epidemiology, has formed the basis for more advanced mathematical models of immune systems representing and describing networks. Epidemic based models to explore several defense strategies that take into account network connectivity as well as the comparison of passive defenses with active defenses in customer networks is discussed in [12]. A continuous epidemic model is used to consider effectiveness in terms of the total number of protected hosts, the total consumed network bandwidth, and the peak scanning rate. These equations include a stochastic parameter which reflects the scanning rate of an infected host and the mean probability of selecting a certain address [13]. A model which takes into account variable infection rates and variable cure rates to take into account the rate and pattern of infection, the network topology, and human countermeasures is discussed in [14]. Discrete mathematical equations that analytically model virus propagation in any network including real and synthesized network graphs are formulated. A general epidemic threshold condition that applies to arbitrary graphs is considered as well as an epidemic threshold model which subsumes many known thresholds for special-case graphs (e.g., Erdos-Renyi, BA power-law, homogeneous). The threshold tends to zero for infinite power-law graphs [15]. Wang et al. [16] present the analysis of epidemiologic models for computer networks including the Kephart-White model [3, 4], the Staniford et al. model [17], the Pastor-Satorras et al. model [18-23], and the Barabasi et al. model [24]. It was pointed out that studies carried out by Kephart and White are based on topologies that do not represent modern networks, Staniford et al. reported a study of Code Red propagation, but did not attempt to create an analytic model, the more recent studies by Pastor-Satorras et al. and Barabasi et al. focused on epidemic models for power-law networks. The simulation and examination of several key characteristics of infection, including the rate of infection through the network and the rate at which individual nodes are re–infected during an attack as well as the theory that the effect of immunization is only effective on certain nodes in the network is discussed in [25]. The Analytical Active Worm Propagation model (AAWP) utilizes a discrete time model and deterministic approximation to describe the spread of active worms [31]. It characterizes the behavior of worms that use random scanning techniques. The model is capable of the characterization of the spread of active worms. The AAWP model is also extended to characterize the spread of a worm that utilizes local subnet scanning. The AAWP model considers the patching rate and the time taken to infect a machine. The first commercial-grade immune system that can find, analyze, and cure previously unknown viruses faster than the viruses themselves can spread is discussed in [30]. A worm simulation model is developed to model the large-scale spread dynamics of a worm and its effects on the network. A homogeneous deterministic epidemic model and a stochastic epidemic model are considered. A spatial epidemic model in order to study scan traffic flows is also discussed in [26]. The space of worm containment systems using reaction time, containment strategy, and deployment scenario and the use of containment mechanisms, content filtering, and blacklisting, in order to control network epidemics are discussed. A continuous model is developed for simulating worm growth and worm containment systems [32]. The two-factor model which is more suited to modeling internet worms as classical epidemic models is developed. Worm propagation is modeled as a continuous differential equation and simulation is done in discrete time. Recursive filtering algorithms for stochastically dynamic immune computing systems are discussed in [33].

The development of a mathematical model of the immune system by investigating immune system processes and optimizing the immune system response is the first step in creating an

immune response to virus attack. Currently, in the study of immune system control, there exist a number of methods connected to processes of optimization intended to solve several problems including immunotherapy and immuno-correction. These methods are also involved in stimulation of the immune system and its distinct cellular populations. The given problems are frequently treated within the framework of optimal control synthesis with respect to chosen performance criterion. The optimal control approach is used in the following problems and applications. The therapeutic enhancement of innate immune response to microbial attack is addressed as the optimal control of a dynamic system [34]. In the model immune response is augmented by therapeutic agents that kill the pathogen directly, that stimulate the production of plasma cells or antibodies, or that enhance organ health. Therapeutic enhancement of humoral immune response to microbial attack is addressed as the stochastic optimal control of a dynamic system [35]. Without therapy, the modeled immune response depends upon the initial concentration of pathogens in a simulated attack. Immune response can be augmented by agents that kill the pathogen directly, that stimulate the production of plasma cells or antibodies, or that enhance organ health. Using a generic mathematical model of immune response to the infection (i.e., of the dynamic state of the system), previous papers demonstrated optimal open-loop and neighboring-optimal closed-loop control solutions that defeat the pathogen and preserve organ health, given initial conditions that otherwise would be lethal [34,36]. Therapies based on separate and combined application of the agents were derived by minimizing a quadratic cost function that weighted both system response and drug usage, providing implicit control over harmful side effects. Optimal control for a class of compartmental models in cancer chemotherapy is discussed in [37]. Optimal control in a model of dendritic cell transfection cancer immunotherapy is explored in [38]. An optimal Human Immunodeficiency Virus (HIV) treatment by maximizing immune response is discussed in [39, 40, 41, 42, 43]. A general mathematical model for cancer chemotherapy as an optimal control problem for a bilinear system is considered. The necessary and sufficient conditions for strong local optimality of bang-bang controls are developed. These results apply to a 3-compartment model which besides a killing agent also includes a recruiting agent. For this model it is shown that singular controls are not optimal, in fact singular regimes for the killing agent are locally maximizing with many optimal bang-bang trajectories near the non-optimal singular arc [44, 45]. A mathematical model for the dynamics between tumor cells, immune-effector cells, and the cytokine interleukin-2 (IL-2) is developed. An optimal control strategy is developed in order to maximize the effector cells and the interleukin-2 concentration and to minimize the tumor cells using numerical techniques [46].

More recently, the concept of an active defense mechanism for a computer network attacked by a worm was presented by Liljenstam and Nicol in [31], where authors discussed different models of active defenses and their effect on the network throughput. The active immune-like network response was called *reactive antibody defense* in [32]. The consideration of active defense mechanisms has immediately prompted the consideration of the effect of limited network resources, such as bandwidth. It was pointed out that the deployment of a so-called anti-worm may significantly consume network bandwidth as well as in the case of a malicious worm [33]. In [33] authors emphasize the importance of applying optimal reactive response that takes into account the infection and treatment costs in terms of network resources. Authors discuss possible ways to determine optimum level of the defense efforts to be applied for a given rate of infection spread that would minimize some total cost function.

A new generation of defense mechanisms for computer networks, minimizing the need for human intervention, became increasingly popular. Authors in [26] presented a technique for automatic generation of an anti-worm through detecting and substituting payload of the malicious worm. As a result, they proposed a method that has a potential for transforming a malicious worm into an opposing anti-worm. A system for automatic revealing susceptible points and generating a patch for target application is presented in [27]. A feasibility of automatic signature generation for worms perpetrating buffer overflow attacks has been studied in [28], [29].

The problem statement for the optimal control of biological systems is formulated as minimization of some criteria that represents the purpose of the immune system to the best understanding of the researcher. For example, the problem of cancer chemotherapy has been formulated [37, 47-49] as an optimal control problem over a fixed interval *[0; T]* with objective given in Bolza form as

$$J(u) = rN(T) + \int_0^T u_1(t)dt \to \min, \qquad (1)$$

where $r = (r_1, ..., r_n)$ is a row-vector of positive weights and the penalty term *rN(T)* gives a weighted average of the total number of cancer cells at the end of the fixed therapy interval

[0; *T*]. The control u_1 - signifies the immune fighter agent. The number of cancer cells which are killed and thus do not undergo cell division at time *t* is given by the portion $u_1(t)$ of the outflow of the last compartment, i.e. $u_1(t)$ is proportional to the fraction of ineffective cell divisions. Since the drug kills healthy cells at a proportional rate, the control $u_1(t)$ is also used to model the negative effect of the drug on the normal tissue or its toxicity. Thus the integral in the objective models the cumulative negative effects of the killing agent in the treatment. Side effects of blocking and recruiting agents are ignored in this formulation. The objective functional is the effect of the immunotherapy while minimizing the cost of the control [47]:

$$J(u) = \int_0^T \left(x(t) - y(t) + z(t) - \frac{1}{2} B(u(t))^2 \right) dt \to \max, \qquad (2)$$

Here the amount of effector and Interleukin-2 cells are minimized while the number of tumor cells and the cost of the control are maximized. *B* is a weight factor that represents a patient's level of response to the treatment. The optimal therapeutic protocol [44] is derived by minimizing a treatment cost function *J*, that penalizes large values of pathogen concentration, poor organ health, and excessive application of therapeutic agents over the fixed time interval, $[t_0, t_f]$ and at the end of the treatment interval. The objective functional to be maximized is [43]

$$J(u_1, u_2) = \int_0^{T_f} \left[T - \left(A_1 u_1^2 + A_2 u_2^2 \right) \right] dt \to \max, \qquad (3)$$

The first term represents the benefit of T cells and the other terms are systemic costs of the drug treatments. The positive constants A_1 and A_2 balance the size of the terms, and u_1^2, u_2^2 reflect the severity of the side effects of the drugs. When drugs such as interleukin are administered in high dose, they are toxic to the human body, which justifies the quadratic terms in the functional. The goal is to maximize the number of T cells and minimize the systemic cost to the body.

Classical methods such as the principle of the Pontryagin [50, 51] and Bellman dynamic programming [52, 53] are used to solve the problems associated with the optimal control of biological system immune response [34-49]. However, the powerful mathematical methods of modern control theory such as dynamic programming can be applied to solve the stated optimization problems even in the case when other methods fail due to computational complexity. It directly involves tasks which entail the control synthesis problem or optimization problems for computer networks [52, 53]. The method of dynamic programming is used in research and in the solution of problems of optimal control for processes in conditions of uncertainty which are typical for computer networks.

In the case of immune system models of computer networks, the problems of optimal response synthesis were solved, according to the problem statement, through defining an optimal trajectory of virus attack suppression using anti-virus programs with respect to different initial conditions [3, 4, 10, 11] in the absence of external disturbance. However, in recent results [54], Pontryagin's Principle was used in finding the optimal control solution for immune system based network models by analogy with the cost optimization problem of Goldman and Lightwood [55] which consisted of minimizing the value of costs incurred from both disease and treatment in biologic epidemiological models. The first cost to be minimized is the cost of infection which is represented by the average delay caused by reduced system performance and increased network traffic. The other cost to be minimized is the cost of treatment which can be represented by node delay due to recently increased filtering. This model [55] uses nonlinear differential equations to provide a qualitative understanding of viral spread.

In the presented work the main principles and mechanisms of immune system response are utilized to solve the problems concerning the development of an optimal computer network immune system response for defense against attacks by malicious software. The main feature of the work is the design and the careful substantiation of a mathematical model of the immune response of a computer network including a description of the dynamics of change of the number of infected hosts, the number of detected and cured computers with anti-virus programs, as well as the dynamics of change of resources as a degree of computer network bandwidth loading under activity of viruses and anti-virus software. The mathematical model is to be constructed according to the combined influence of virus attacks and anti-virus software, and also their direct influence on computer network resources.

In 2001-02 the authors were engaged in the project *BASIS (Biological Approach to System Information Security)* funded by Air Force Research Laboratory. This effort was aimed at the establishment of important similarities between a biological immune system and a computer network subjected to an information attack that could be explored for the development of the next generation of computer network defenses. This project resulted in a number of publications [1], [2] and provided the team with valuable new concepts in computer security.

Project *Recognition of Computer Viruses by Detecting their Gene of Self-Replication*, also funded by the air Force Office of Scientific Research (AFOSR), has been exploring the notion that while most malicious software self-replicates in order to create a computer epidemic maximizing its destructive effect, self-replication of legitimate software is very uncommon. At the same time, the number of practical self-replication techniques utilized in viruses and worms is quite limited and requires the developers of new attacks to utilize the same "old" self-replication techniques in new viruses and worms. Consequently, the detection of self-replication functionality in computer code provides the basis for the detection of both known, and what is more important, previously unknown malicious software. It was found that monitoring and analysis of system calls during the execution time provides the most dependable approach for the detection of attempted self-replication [56]. This has resulted in the development of a Dynamic Code Analyzer (DCA), a resident software tool that monitors system calls and detects specific subsequences (patterns in the system call domain) indicative of self-replication. The process engaged in self-replication would be suspended and the user is given an authority to continue or terminate the process. The DCA has been successfully tested against both known and previously unknown malicious software. It could be seen that while DCA may not prevent the damage caused to an individual host, it surely prevents the development of computer epidemics.

It is expected that a self-replicating anti-worm would implement the most advanced propagation strategies resulting in disinfection and/or immunization of individual hosts. Consequently, both the worm and anti-worm activity has to be quantified by the number of infected hosts and the number of disinfected or immunized hosts correspondingly. Worm activity has a strong impact on the bandwidth of networks. The bandwidth of a network can be considered as the most relevant network resource that affects both the quality of network operation as well as the propagation of self-replication software. Consequently, the bandwidth of a network becomes a key factor to be addressed in a mathematical model of the network's immune response. While there are many alternative ways to quantify the bandwidth of a network, it typically represents the amount of transmitted information per unit of time.

The resultant AIR model follows the principles of operation of a biologic immune system describing the interaction between resources of the organism, antigens, and the immune system. The model variables represent (1) network *resources*, expressed as the available capacity (bandwidth) of the information channels (bits/sec) utilized by the worm and anti-worm software, (2) number of disinfected/ immunized hosts, and (3) number of infected hosts. It could be seen that the above variables are interrelated: differential increments of infected and disinfected hosts directly affect the network bandwidth, propagation rates of the worm and anti-worm depend on the available network bandwidth, the number of infected computers depends on the propagation rates of worm and anti-worm, etc. The developed model comprises several nonlinear stochastic differential equations; the stochastic nature of the AIR model is reflected by the fact that the numbers of infected and disinfected/immunized hosts are statistical estimates of the respective quantities that in reality could be obtained by the scanning of a relatively small group of randomly chosen hosts (selective sampling).

In summary, the following critical features of the active immune response could be noted:

- Distributed detection/identification mechanism
- Synthesis of highly specialized defense agents on demand
- Controlled self-replication of the defense agents

- Efficient status assessment of the network
- Negative feedback control assuring sustainable operation of the network

The implementation of the active immune-type response in a computer network security system provides the only alternative in the assurance of dependable network operation of in the nearest future. This could be achieved by the development of a fully automatic computer network security system capable of timely detection and mitigation of information attacks perpetrated by self-replicating malicious software is proposed. The system will detect an attack and synthesize and deploy specialized self-replicating anti-worm software for attack mitigation. It will require an advanced feedback control scheme to insure the observability and controllability of the overall process thus preventing the overload of the network bandwidth will be implemented. Efficient procedures for attack detection, feedback generation and control of self-replication of the anti-worm are to be developed.

3.0 MODELING THE IMMUNE-TYPE RESPONSE OF A COMPUTER NETWORK

A mathematical model provides the basis for the implementation of various control schemes facilitating the deployment of an "observable and controllable" anti-worm within the limited bandwidth of the network thus achieving sustainable operation of a network subjected to an information attack.

Mathematical modeling the immune response necessitates analyzing the factors responsible for the propagation of viruses and revealing the most significant of them in order to define strategies which have a high probability of being used by the creators of worms in the future.

Developers of malicious software are constantly deploying new worms and optimizing their distribution speed. Nowadays there are two types of viruses; those which effectively work in the Internet and those which work in a local area network. Quickly propagating Internet worms such as Code Red and SQL Slammer are much less effective in a local area network than viruses using local addressing such as W32.Blaster. It is possible to create a hybrid virus which will randomly choose a network, and consequently scan it.

Scanning a network can be performed by scanning via a predetermined "hit list" or scanning by permutation. When scanning from a hit list, the virus selects only a few computers from a list of thousands of vulnerable IP-addresses. Permutable scanning is performed in a random fashion making sure that IP addresses are not scanned more than once.

It should be noted that Slammer did not use either of these two types of scanning to achieve its fast and massive world wide propagation. This worm used random scanning only. In other words the worm used the Pseudo Random Number Generator (PRNG) to generate a list of IP addresses which were used to scan for hosts accessible via UDP port 1434. After the detection of such computers, the worm sends them its shell code in an attempt to infect them and continues scanning for vulnerable machines.

Although random scanning is considered to be ineffective in comparison with other IP address generating techniques, it is still a fast enough technique to be extremely devastating. In fact it was so impetuous, that the majority of network administrators were unable to respond to the worm before their systems became infected.

The model results of [15] demonstrate how different scanning strategies influence the total speed of virus propagation. According to the given results, the factors having the greatest influence on a simulated distribution of a network virus have been revealed as follows:

i) Address sampling: The method of address sampling has a huge impact on the speed of virus propagation. Various methods can include completely random sampling, random sampling with local preferences, and consecutive polling.
ii) Multithread processing: Scanning in a single thread of the process is much lower in many duplicated threads.
iii) Method of preliminary scanning: Defining hosts by "listening" to the required port before sending data to the hosts boosts the effectiveness of the worm.

In current network immune systems, [12, 17-23], mathematical models are described by stochastic equations and are characterized as Markov processes. The following discussion constitutes the basis of the work and is intended to consider several variants of the presented models.

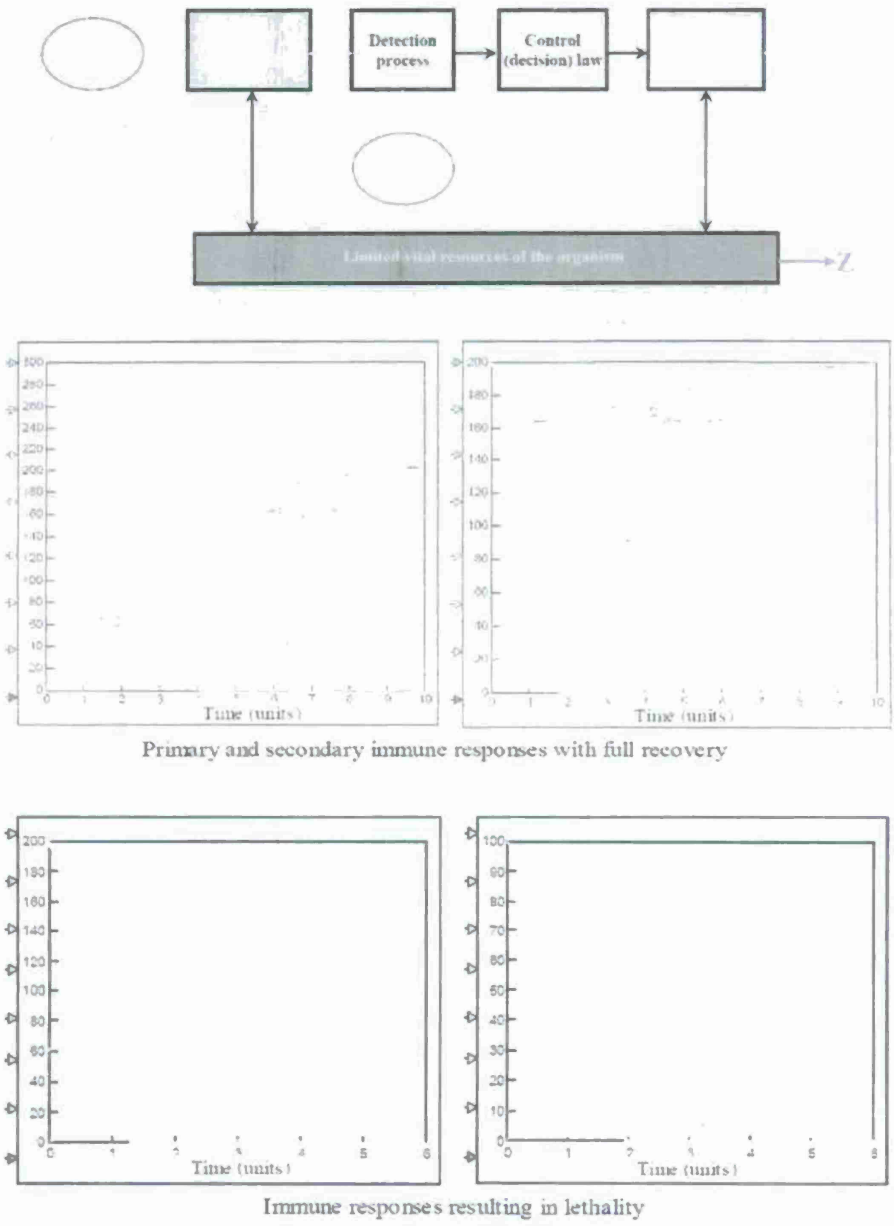
In order to propagate, computer worms scan a wide-area network, striving to find vulnerabilities in software of individual hosts. Having found the IP-address of a vulnerable host, the worm simply injects itself from the network into the unprotected machine using previously revealed vulnerabilities. Once installed in the host's memory, the worm, searches for other vulnerable computers and sends itself to their IP addresses. While different types of worms use different propagation strategies and exhibit different propagation rates, all of them successfully propagate, using the slightest vulnerability to penetrate existing security systems. It could be seen that the number of infected (or scanned) hosts is one of the variables quantifying the attack.

Worm activity has a negative impact on the bandwidth of networks. For example, the code size of the quickly propagating Slammer is very small, 404 bytes including the header. The small size accounts for the high speed of its propagation. During the first minute of attack, the quantity of infected computers grew exponentially, doubling every 8.5 seconds. By searching for potential victims during a 10-minute time period, the worm scanned about 3.6 billion out of approximately 4 billion existing Internet addresses, reaching the scanning rate of 55 million hosts per second during the first three minutes of the attack. As the attack progressed, this rate decreased due to the limited networks' bandwidth. Although the worm did not carry destructive instructions, as did Code Red and Nimda which changed files and damaged web-sites, it caused serious damage during the peak of the epidemic consuming a significant portion of the Internet bandwidth, and interfering with the operation of many servers. Consequently, the bandwidth constitutes the main network resource relevant to our problem, and properly quantified presents another important characteristic of the attack.

A mathematical model of the network response cannot be established without describing the effects of anti-worm activity, i.e. the worm-like propagation of anti-worm programs. Such anti-worm technology could be exemplified by Welchia, which is one of a few worms intended for the neutralization of another malicious program, Blaster worm. In the same way as Blaster, Welchia penetrates into a computer through a gap in Windows firewall, first having verified that the computer is infected with Blaster. Welchia then deletes Blaster, completely restores the attacked system, and loads the Microsoft update thereby fixing the vulnerability. Understandably, the propagation of anti-worms is affected by the limited networks' bandwidth and can be described by the number of disinfected (or scanned) hosts or released anti-worm software units.

Now it could be seen that the mathematical model of the network's immune-type response should describe the complex, dynamic interaction of three factors, the number of infected hosts (or scanned by the worm), the number of disinfected hosts (scanned by the anti-worm, or released anti-worm software units) and their combined effect on the network bandwidth. Following the principles of operation of the immune system describing the interaction between resources of the organism, antigens, and the immune system, we bring into consideration the basic variables of the mathematical model establishing the correspondence between the "biological" and "computer network" concepts. Figure 2 below provides a block diagram of a

computer model of active immune-type response and simulation results reflecting various initial conditions, severity of infection, level of the resources of the organism, and level of immunity of the organism (i.e. previous exposure to the same attacker).



Primary and secondary immune responses with full recovery

Immune responses resulting in lethality

Figure 2: Block Diagram of a Mathematical Model of Active Immune-type Response and Simulation Results Corresponding to Different Initial Conditions

3.1. Continuous-time Model

System resources: The bandwidth represents the joint capacity of the network's communication channels and could be expressed in bit/sec. During the active network response the bandwidth is utilized by both the worm and anti-worm. Introduce variable $W(t)$ that denotes the bandwidth consumed by the worm and anti-worm during the attack. Note that before or after attack this variable exhibits all properties of a characteristic of a stable, inertial system and exponentially returns to the value of zero with time (network "inertia" accounts for the finite connectivity within the network, its distributed nature, and various phenomena slowing its operation). The equation describing the natural dynamics of the network bandwidth in the absence of attacks could be written as:

$$\frac{dW(t)}{dt} + \gamma W(t) = 0, \quad W(0) = W_0 \tag{4}$$

where $\gamma > 0$ represents the "natural" rate of change of bandwidth, and $W_0 \geq 0$ is some initial value.

Number of infected hosts: Let variable $X(t)$ represent the number of infected computers that would exponentially increase due to the worm propagation and decrease due to the anti-worm activity. The equation describing the dynamics of the number of infected computers at the very early stage of the attack could be described as:

$$\frac{dX(t)}{dt} - \alpha X(t) = 0, \quad X(0) = X_0 \tag{5}$$

where $\alpha > 0$ is the "natural" rate of change of quantity $X(t)$, X_0 is some initial value characterizing the intensity of the attack.

Number of deployed units of anti-worm software: Let $Y(t)$ represent the number of deployed units of anti-worm software. It is expected that this number, governed by some control mechanism, will be exponentially increasing to oppose the propagating worm and decreasing as the worm if being defeated. The natural dynamics of the appropriate process at the early stage of the attack can be defined as:

$$\frac{dY(t)}{dt} + \beta Y(t) = 0, \quad Y(0) = 0 \tag{6}$$

where $\beta > 0$ represents the "natural" rate of change of variable $Y(t)$.

Now let us consider the interaction of the above variables during the attack. The network bandwidth during the attack will be consumed by scanning activities performed by worm and anti-worm, and file transmission causing infection and disinfection of computers. To quantify these effects, equation (4) is extended as:

$$\frac{dW(t)}{dt} + \gamma W(t) = \theta X(t) + \delta Y(t) \qquad (7)$$

where $\theta X(t)$ and $\delta Y(t)$ represent the bandwidth consumption by the propagating worm and anti-worm correspondingly, θ and δ are positive constants.

Reconsider equation (5) taking into account that the bandwidth consumption slows the worm propagation process and the anti-worm activity can reverse it. This reality could be described as follows

$$\frac{dX(t)}{dt} - [\alpha - \lambda W(t) - \mu Y(y)] X(t) = 0 \qquad (8)$$

where λ and μ are positive proportionality coefficients.

Now let us modify equation (6) accounting for the effect of the changing network bandwidth on the propagation of the anti-worm and assuming that the anti-worm generation effort is proportional to the number of infected computers (simple proportional control law). Then the number of deployed units of the anti-worm software will be changing according to the equation:

$$\frac{dY(t)}{dt} + [\beta + \varphi W(t)] Y(t) = \rho X(t) Y(t) \qquad (9)$$

where ρ is shows the rate of detection of the infected hosts. Note that the product $\rho X(t)Y(t)$ signifies the fact that as the number of infected hosts decreases, it becomes more difficult to account for them by the control activity.

Thus, *the mathematical model of the immune-type response of a computer network* is described by the following system of continuous nonlinear differential equations, describing the inertial properties of the network and complex interaction between its key variables, the increment of the network bandwidth, the number of infected hosts, and the number of deployed units of the anti-worm software:

$$\begin{cases} \dfrac{dW(t)}{dt} = -\gamma W(t) + \theta X(t) + \delta Y(t) \\ \dfrac{dX(t)}{dt} = (-\lambda W(t) + \alpha - \mu Y(t)) X(t) \\ \dfrac{dY(t)}{dt} = (-\varphi W(t) + \rho X(t) - \beta) Y(t) \end{cases} \quad \text{or} \qquad (10)$$

$$\begin{bmatrix} \dot{W}(t) \\ \dot{X}(t) \\ \dot{Y}(t) \end{bmatrix} = \begin{bmatrix} -\gamma & \theta & \delta \\ -\lambda X(t) & \alpha & -\mu X(t) \\ -\varphi Y(t) & \rho Y(t) & -\beta \end{bmatrix} \begin{bmatrix} W(t) \\ X(t) \\ Y(t) \end{bmatrix} \qquad (11)$$

with the initial conditions

$$W(0)=0,\ \ X(0)=X_0>0,\ \ Y(0)=Y_0>0 \qquad (12)$$

Analysis of the above equations indicates that they represent the natural motion of a nonlinear system that in the case of global asymptotic system stability and regardless of the initial severity of the attack, signified by the value of X_0, results in

$$W(\infty)=0,\ \ X(\infty)=0,\ \ Y(\infty)=0, \qquad (13)$$

i.e. full recovery of the network. It could be seen that the above assumption is not realistic unless the "sense of reality" is assured by the condition

$$W(t)\le W^{MAX}=\frac{\alpha}{\lambda} \qquad (14)$$

thus preventing the full congestion of the network communication channels.

It is important to realize that equations (10, 11) are stochastic by their very nature: their parameters reflect the combined behavior of numerous hosts of the network and therefore the obtained model can reliably describe the network in general but not any individual host. Moreover, numerical values of the model parameters reflect properties of a particular network and are subject to change depending on many factors including the propagation engine and payload of the particular worm.

Figure 3 presents the simulation setup implementing the above model in the SIMULINK environment.

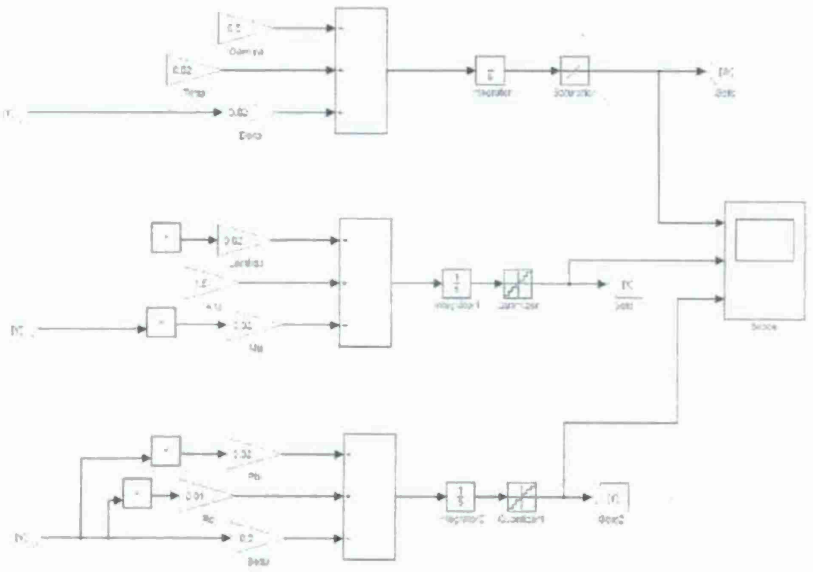


Figure 3: Simulation Setup

The results of the simulation experiment (Fig. 4) indicate that with the appropriate choice of the model parameters, it realistically describes the nature of the active network response (in relative units): the increase of the number of infected hosts from some initially introduced value, increase of the number of deployed anti-worm software units in response to the attack, drop of the bandwidth resource of the network due to the worm and anti-worm propagation, and finally, the full recovery of the network from the attack.

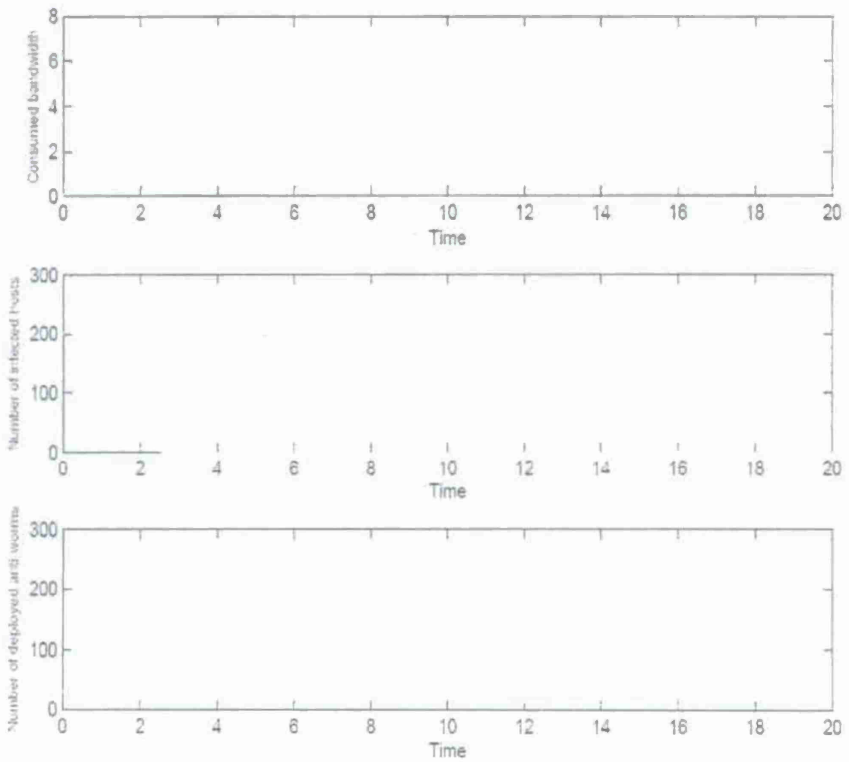


Figure 4: Propagation Dynamics of the Worm and Anti-worm During Active Network Response

3.2. Discrete-time Model

The discrete-time model comprises the following system of equations:

$$\begin{cases} X(i) = \alpha_i \cdot n_w \cdot X(i-1) \cdot \delta_i - k_i \cdot Y(i-1) \cdot \delta_i \cdot \gamma_i + X(i-1) \\ Y(i) = Y(i-1) + k_i \cdot Y(i-1) \cdot \delta_i \cdot \beta_i - \eta \cdot Y(i-1) \\ W(i) = -W(i-1) \cdot \psi + X(i-1) \cdot n_w \cdot \theta + k_i \cdot Y(i-1) \cdot \theta \end{cases} \quad (15)$$

where

$X(i)$ is the number of infected hosts,

$Y(i)$ is the number of anti-worm instances in the network, and

$W(i)$ is the amount of bandwidth consumed by the worm and anti-worm activity.

The parameters of the model are described in the Table 2 below.

Table 2. Model Parameters

Parameter	Formula	Description
α_i	$\alpha_i = (N - X(i-1) - Y(i-1))/N$	Probability of hitting a vulnerable host at time i (not yet infected by the worm, nor the anti-worm)
δ_i	$\delta_i = (1 - W(i-1)/W_{max})$	Packet delivery probability
k_i	$k_i = X(i-1) \cdot \mu$	Rate of sending attack packets by every unit of anti-worm at time i
β_i	$\beta_i = (N - Y(i-1))/N$	Probability of hitting a vulnerable host or a host infected by the worm at time i (a victim of the anti-worm)
γ_i	$\gamma_i = X(i-1)/N$	Probability of hitting a host infected by the worm
μ	Constant gain	Proportional coefficient used in the anti-worm generation rate control law
n_w	Constant gain	Rate of sending attack packets by every unit of the worm
θ	Constant gain	Amount of consumed bandwidth due to transmitting of one attack packet of a worm or an anti-worm
ψ	Constant gain	Coefficient determining the rate of decreasing of the amount of consumed bandwidth (in natural motion)
N	Constant gain	Total number of susceptible hosts in the network (before worm and anti-worm activity)
W_{max}	Constant gain	Maximum available bandwidth
η	Constant gain	Rate of the anti-worm population decrease in natural motion

As model (15) was developed, several reasonable assumptions were made and simplifications were introduced. First, the probabilities α_i, β_i and γ_i were approximated assuming an identical and independent distribution. We neglected network topology and infrastructure limitations. Also, for the sake of simplicity, we assumed that the worm and anti-worm have only one packet attack and utilize a QoS free transport protocol such as UDP. For an example of such a worm in real life, one can refer to the well-known SQL.Slammer worm. Moreover, we assumed that network natural activity is much smaller than the worm and anti-worm combined activity. These simplifications allowed us to use the linear law for packet delivery probability. While the linear formula may not capture network congestion phenomena in all phases, it is a good choice to consider the network bandwidth as a single value.

We performed our simulation in MATLAB using the static parameters and initial values listed in Table 3.

Table 3. Model Static Parameters

μ	0.00005
n_w	0.01
θ	0.5
ψ	0.02
N	10000
W_{max}	240
$X(0)$	200
$Y(0)$	10
η	0.000001

Figure 5 shows the consumed bandwidth as a percentage of maximum bandwidth, the number of infected hosts and the anti-worm propagation rate. One can see that anti-worm generation rate rapidly increases until the worm population drops down and then slowly decreases along with the worm population size.

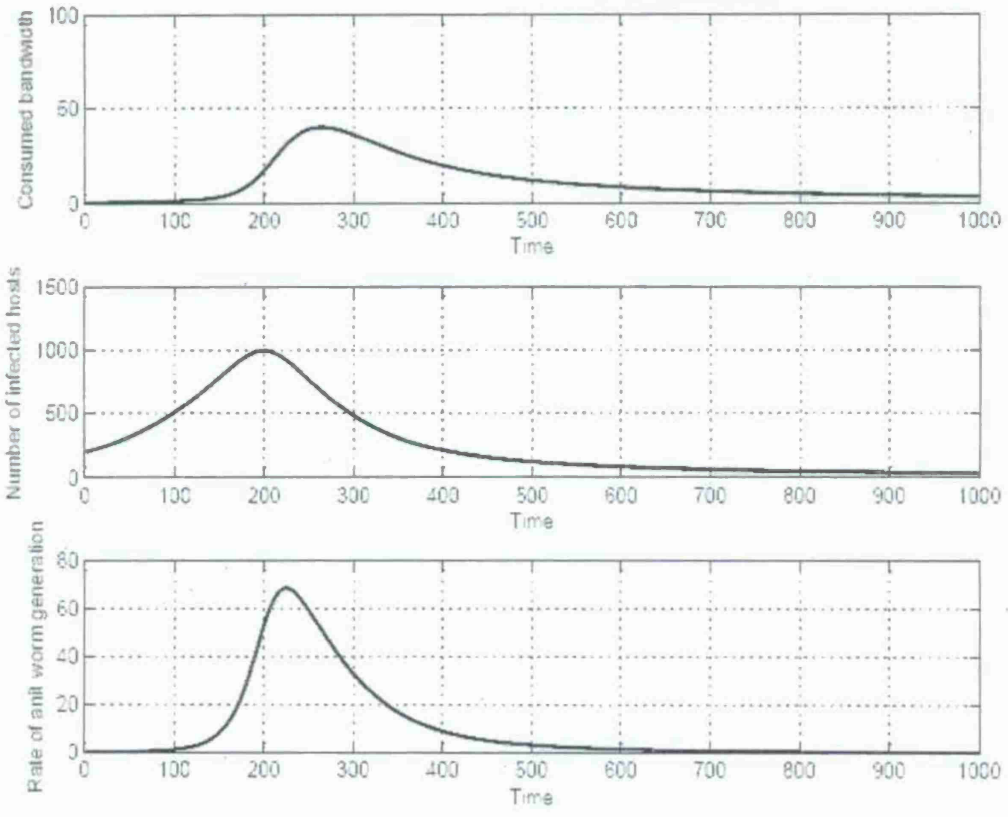


Figure 5: Worm and Anti-worm Dynamics

The value of the obtained mathematical models is in the demonstrated ability to describe the interaction between the major variables describing the behavior of a network equipped with anti-worm defenses under the attack. Consequently, the model provides a basis for the development and validation of advanced feedback control mechanisms of the network defenses.

4.0 AUTOMATIC DETECTION/MITIGATION OF COMPUTER WORM ATTACKS

A successful solution to computer/network security problem implies that a fully automatic defense mechanism emulating the active immune response be developed and deployed in the network. This can be achieved by understanding the complex phenomena of co-existence of adversarial self-replicating entities in the network, worm and anti-worm (virus and anti-virus) in conjunction with their effect of the network bandwidth. While any deployment of self-replicating software could be highly detrimental to the network, it is necessary to develop a technology for making the process of co-replication of worm and anti-worm observable and controllable. The critical features of the active immune response, mentioned earlier, must be implemented:

- Distributed detection/identification mechanism
- Synthesis of highly specialized defense agents on demand
- Controlled self-replication of the defense agents
- Efficient status assessment of the network
- Negative feedback control assuring sustainable operation of the network

This implies that the necessary components of the immune-like defense mechanism are:

- Highly efficient IDS deployed in a number of specially designated machines distributed within the network for the detection and correlation of instances of self-replication behavior of software indicative of computer worm propagation.
- A special station(s) synthesizing on demand anti-worm entities that could be released in the network upon the detection and identification of the attacker. These entities must be strictly specialized thus targeting only the designated malicious processes. The propagation mechanism of the anti-worm should be equivalent to the one of the most efficient worms, or will utilize the list of IPs of the network. The replication (propagation) rate of the anti-worm entities must be subjected to real-time centralized control. A technology facilitating a controlled time-varying rate of self-replication of the anti-worm is proposed.
- A statistical selective sampling procedure is proposed for the estimation of the number of hosts in the network affected by the worm and anti-worm by periodic scanning a relatively small group of randomly chosen hosts. This approach, minimizing the impact on the network bandwidth, is suggested as the means for the generation of a feedback signal that is crucial for the implementation of any feedback control mechanism.
- Finally, an advanced control law relating the propagation rate of the anti-worm to the feedback information has to be established in full compliance with modern control theory.

The principle of operation of an automatic defense mechanism capable of controlled deployment of a self-replicating anti-worm in response to an information attack on a computer network by self-replicating software is shown in Figure 6. This figure depicts malicious software deployed from an attacker's computer as it propagates within the network. Eventually, it is detected by one of the specialized attack detection/identification stations dispersed within the network that automatically extracts a binary signature of the detected malicious software and transmits it to the *Control station*. Then a specialized anti-worm (self-replicating patch) is deployed from the control station and also propagates through the network disinfecting and immunizing individual hosts. Based upon the topology of infected/uninfected hosts the Control station will ether send

self-replicating anti-worm software to prevent the propagation of the worm or logically disconnect hosts from the network.

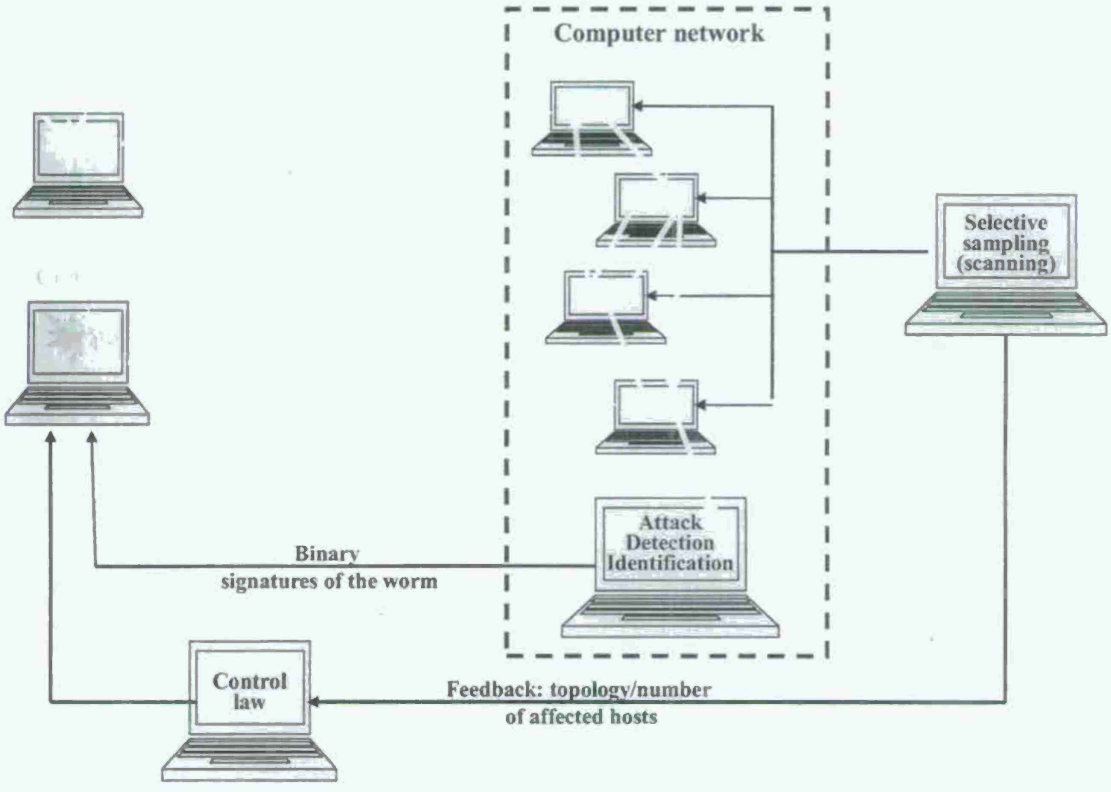


Figure 6: Principle of Operation of Automatic Security System for a Computer Network

The propagation rate of the anti-worm is time-dependent and is defined by a specially designed control law. The anti-worm is targeting the malicious software specified by a binary signature loaded in its targeting mechanism. The network computers (routers) are subjected to periodic selective scanning in order to estimate the number of infected and disinfected (immunized) hosts. The control law converts the feedback information on the overall status of a computer network subjected to both the information attack and the effects of the self-replicating anti-worm into numerical values of some parameters governing the propagation rate of the anti-worm. This control law is synthesized on the basis of nonlinear differential equations describing complex interplay between the number of infected hosts, number of operating anti-worm units, and the available resources of the network – the remaining capacity of its communication channels.

4.1. Attack Detection/Identification

Attack detection/identification stations are interconnected host-based worm detection systems dispersed within the computer network. Such a station is visualized as a cluster of independent virtual computers, vulnerable to attacks, being run on the same physical host. In a way, the virtual computers of the detection stations form a "network within the network" and exchange security related information that is important for reliable detection of the attack and identification

of the attacker. Each virtual computer of the station is equipped with the Dynamic Code Analyzer (DCA) capable of detecting attempted self-replication [56]. When attacked by a computer worm or virus, the virtual computer of the detection station executes the instructions of the malicious software including those resulting in its self-replication. It is commonly known that most malicious software self-replicates in order to maximize the severity of the attack, and this functionality typically invokes specific sequences (patterns) of system calls. The DCA monitors system calls and is capable of detecting the specific patterns of system calls indicative of self-replication. Upon detection of the attempted self-replication, the DCA suspends the process in question, generates a specific warning identifying the detected specific self-replication pattern and transmits it to all virtual machines of the detection station. While the system calls-based IDS are known to have a high rate of false positives, no further action is taken until the same alarm is generated by another virtual machine, thereby drastically reducing the probability of false alarms. When the alarm is substantiated by a warning from an additional source, a special procedure deployed at one of the detection stations, capable of extraction of the binary signature of the troublesome software is activated. The feasibility and general algorithms of such procedures are discussed by several authors [28], [29], [30], [31]. The extracted binary signature will serve as an ID of the worm/virus attacking the network and it will be transmitted to the control station.

One of the following sections of this Report present the authors' research aimed at the enhancement of the IDs technology recommended for this application.

4.2. Generation of the Feedback Signal

A successfully implemented feedback signal is crucial for any self-regulating system. In our situation the feedback signal represents the number of infected and disinfected hosts that varies as the attack and countermeasures progress. While scanning of the individual hosts of the entire network in a worm-like fashion is clearly unacceptable, a selective sampling (scanning) approach commonly utilized in quality control is adopted, and then the resultant problem is solved using statistical inference.

For a given population size and fixed sample size this problem is usually solved under some assumptions about the underlying statistical distribution of the sample that is expected to be geometric. In reality, one does not know the population size, i.e. the total number of hosts, since at any time computers may be arbitrarily connected and disconnected from the network. As a result, the geometric distribution assumption is not truly applicable. However, if the number of susceptible hosts is sufficiently large, we do not have to know the exact number of vulnerable computers to apply the binominal distribution. Presumably, the share of infected computers in a fixed size sample confirms to binominal distribution with the mean value equal to the share of infected computers in the whole population. In order to use statistical inference techniques to estimate the mean value one has to ensure that the sample is *identically and independently* distributed. In other words, the source must be stationary during the sample selecting process that could be assured by two quite realistic assumptions, (1) hosts whose status is polled are chosen uniform randomly, and (2) during the scanning session the number of vulnerable hosts and number of infected machines does not change.

The *independence* requirement can be achieved using a random number generator. The *identical* property can be satisfied only if computers would be scanned pseudo simultaneously, i.e. by a very short duration scanning session.

Under the assumption of binominal distribution, the percentage of infected computers is estimated as a proportion parameter. The required statistical significance of the estimate could be assured by the choice of the sample size. While a closed-form solution for the exact confidence interval by the acceptance region inversion does not exist, approximate solutions for a confidence interval based on Central-Limit Theorem could be considered. Then a Clopper-Pearson (exact) confidence interval for the estimate that provides an assigned coverage probability with any specified precision can be found via standard numerical methods. Its computation results in an iterative process leading in the definition of the minimum sample size providing a specified relative error (statistical significance) of the estimate. Ultimately, this provides a theoretical foundation for the utilization of selective scanning as the means of establishing a dependable feedback representing the dynamics of the worm/anti-worm interaction process in the network without a significant impact on the network bandwidth.

The developed approach to continuous network status assessment is given in the next section of this report.

4.3. The Control Station

A control station is a computer equipped with software for generation and controlled dissemination on demand of a specialized anti-worm (anti-virus). An anti-worm is an already existing software entity that consists of a propagation engine, a targeting mechanism and a payload, see Figure 7. The propagation engine implements the most efficient propagation techniques of computer worms that imply scanning of the network in the search of susceptible hosts and transmitting the appropriate code to such hosts [30], [31]. However, unlike the common worm intended to maximize the effect on the network, the propagation rate of the anti-worm will be controlled according to a control law that takes into account the number of infected hosts and the availability of the network bandwidth.

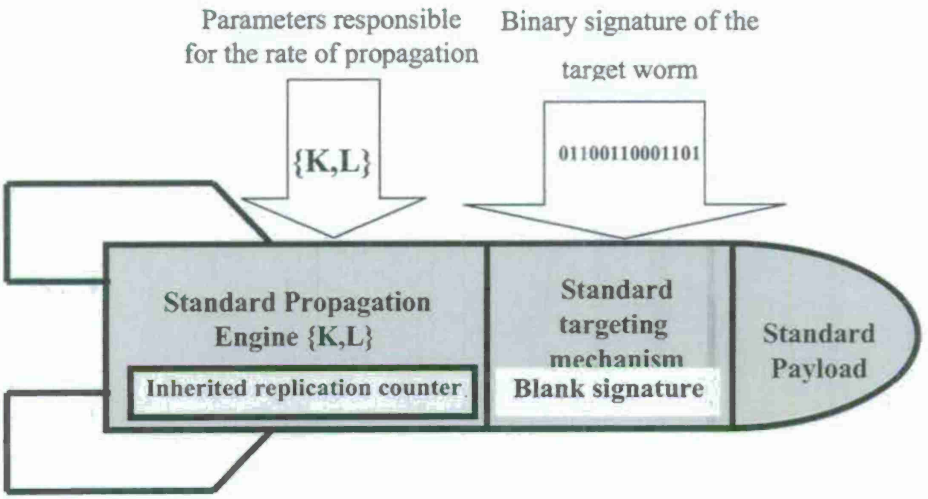


Figure 7: Composition of an Anti-worm

This could be achieved by an inherited replication counter that controls the allowable number of generations, K, and the allowable number of offsprings in each generation, L, as the anti-worm propagates. The K and L numbers are time-dependent and are defined by the control law and communicated to the control station. It could be seen that having such a finite and controllable $\{K,L\}$ combination results in the limited life span of the anti-worm so that it will cease to exist as the attacking worm is defeated, and due to the properly chosen control law does not congest the network.

The standard payload of the worm implements functions resulting in the neutralization of a worm if the host is infected and/or immunization of the host by making it immune to the worm. Both functions could only be performed in response to a particular worm that is uniquely identifiable by its binary signature and detected by the standard targeting mechanism operating in the traditional anti-virus software fashion. This could be achieved by loading the binary signature of the propagating worm extracted by one of the detection stations and transmitted to the control station.

The anti-worm is designed to propagate in the same environment utilizing some of the same vulnerabilities exploited by worms. This strategy is applicable to anti-worm propagation in vulnerable systems, as well as systems that are already infected with the worm. Certain worm propagation scenarios may include a self-patching worm behavior that would prevent an anti-worm from replicating onto already infected machines. This situation may be resolved by implementing such methods as establishing a dedicated communication channel for anti-worm propagation, or disabling self-patching at the system level by utilizing a run-time modification prevention approach. While such counter-measures are less favorable in the global net environment due to local system modification requirements, they would further enhance the functionality of the proposed active immune response system.

It should be noted that this anti-worm propagation approach is the most conservative and the most generic. Technically, anti-worm propagation strategy can take full advantage of the friendly system environment and utilize to known list of IP addresses of the computers within a particular network. Although this would constitute a step away from the "active immune-type response", such an approach seems to be practical in some application problems.

In an effort to speed the recovery of an infected network, the various control stations will also have control of administratively close routers allowing for quarantine and isolation of the infected portions of the network. If the infected portions of the network can be isolated quickly, not only will further spread of the worm be reduced, but the transmission of the anti-worm will be more successful as networking resources will be worm free and available. In addition, knowledge of the infection concentration will aid in guiding the anti-worm propagation.

Several ways of disconnecting or isolating hosts by the control station could be investigated. Possibilities include disconnecting the host from the network by disabling its various network interfaces, altering network address resolution tables, or by dynamically changing host routing information and reconfiguration of network routers and switches. Some of these techniques are rather simple to implement but may not offer much flexibility in the recovery period while others can be quite complex, involving networking hardware from several vendors with very different configuration methods.

The following are some conditions assuring the feasibility of the proposed anti-worm generation technology:

1. The generator of anti-worms must be dependably protected from attacks.

2. Potentially, the anti-worm mechanism can be exploited by real warm (as AIDS does). In that case the immune system is reoriented to generate the warms. This reality places high emphasis of advanced encryption.

3. A special attention should be paid to providing distinctive attributes for anti-worms enabling the system to recognize anti-worms as such.

4. There are two ways of software distribution: remote one by download and local - by setup. The advantages of the local way of anti-worm distribution (more difficult to compromise) must be assessed against the remote distribution (more consistent with the concept of active immune-type response) must be analyzed. Unfortunately, such a study has not been performed under this project.

4.4. The Control Law

The control law is responsible for the stability and efficient operation of the described system. It defines a relationship between the control parameters of the anti-worm $\{K, L\}$ responsible for the rate of its propagation, and the feedback information representing the severity of the attack, namely, the estimated number of infected hosts, disinfected hosts and the available network bandwidth. The control law is formulated on the basis of the mathematical model of the active network response featured in Section 3 of this Report. After the expansion, linearization and

conversion into the discrete-time domain of equations (10, 11), this model could be written in a traditional discrete-time state-variable notation as

$$\begin{bmatrix} w(n+1) \\ x(n+1) \\ y(n+1) \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} w(n) \\ x(n) \\ y(n) \end{bmatrix} + \boldsymbol{B} \begin{bmatrix} \boldsymbol{K}(n) \\ \boldsymbol{L}(n) \end{bmatrix}, \quad \begin{bmatrix} \boldsymbol{K}(n) \\ \boldsymbol{L}(n) \end{bmatrix} = -\boldsymbol{F} \begin{bmatrix} w(n) \\ x(n) \\ y(n) \end{bmatrix} \tag{16}$$

where

$w(n)$, $x(n)$ and $y(n)$ are state variables of the active network response that correspondingly represent the increment of the network bandwidth, number of infected hosts, and number of active (i.e. engaged in scanning activities) anti-worm units at particular moments of discrete time, $n = 1,2,\ldots$

$\boldsymbol{K}(n)$ and $\boldsymbol{L}(n)$ are the control efforts of the active network response representing the allowable number of generations and the allowable number of offsprings in each generation of the anti-worm,

$\boldsymbol{A}$ is a matrix representing linearized complex interrelationships between the number of infected hosts, number of active anti-worm units and the network bandwidth,

$\boldsymbol{B}$ is a matrix representing linearized effects of the control parameters of the anti-worm on the number of infected hosts, number of active anti-worm units and the network bandwidth,

$\boldsymbol{F} = \boldsymbol{F}(n)$ is a matrix of the state-variable controller,

A combination of the matrix-vector equations written above results in the discrete-time description of the closed-loop system that effectively is the equation of natural motion of an inertial system

$$\begin{bmatrix} \mathrm{w(n)} \\ \mathrm{x(n)} \\ \mathrm{y(n)} \end{bmatrix} = (\boldsymbol{A} - \boldsymbol{BF}) \begin{bmatrix} \mathrm{w(n)} \\ \mathrm{x(n)} \\ \mathrm{y(n)} \end{bmatrix} \tag{17}$$

It is known that the steady-state regime of a dynamic system described by such an equation is dependent on the eigenvalues of matrix $\boldsymbol{A}^{CL} = \boldsymbol{A} - \boldsymbol{BF}$ and such a system is stable, i.e. all eigenvalues of matrix $\boldsymbol{A}^{CL}$ are located in the left-hand half of the complex plane, is

$$\begin{bmatrix} \mathrm{w}(\infty) \\ \mathrm{x}(\infty) \\ \mathrm{y}(\infty) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

Therefore, it could be seen that for arbitrary matrix A and matrix B appropriate selection of the controller matrix F would assure that the network engaged in active response returns to its status preceding the attack. Moreover, the selection of specific eigenvalues of matrix ACL utilizing eigenvalue assignment techniques common in modern control theory enables the designer to manipulate the duration of the entire attack mitigation process.

The authors are aware of the fact that the above situation is overly optimistic due to the fact that matrix A and matrix B are poorly known and their values are time-dependent due to changing properties of the network and moving point of linearization. It is proposed to estimate these matrices experimentally by periodic deployment of a specially designed short-lived worm. The poor knowledge of matrices A and B and their time-dependence could be addressed by the application of a model-reference (adaptive) control law that results in a time-dependent controller matrix $F = F(n)$. The resultant control system configuration is depicted in Figure 8.

Figure 8 shows a reference model (a simulation module) representing the required closed-loop dynamics of the active response assured by the selection of its fundamental matrix A^M,

$$\begin{bmatrix} z_1(n+1) \\ z_2(n+1) \\ z_3(n+1) \end{bmatrix} = A^M \cdot \begin{bmatrix} z_1(n) \\ z_2(n) \\ z_3(n) \end{bmatrix} \tag{19}$$

It could be seen that the state variable of the active response, $x(n)$, representing the number of infected hosts and the corresponding state variable of the reference model, $z_1(n)$, have the same initial conditions, $x(0) = z_1(0) = x_0$ equal to the initial number of infected hosts estimated at the detection of the attack.

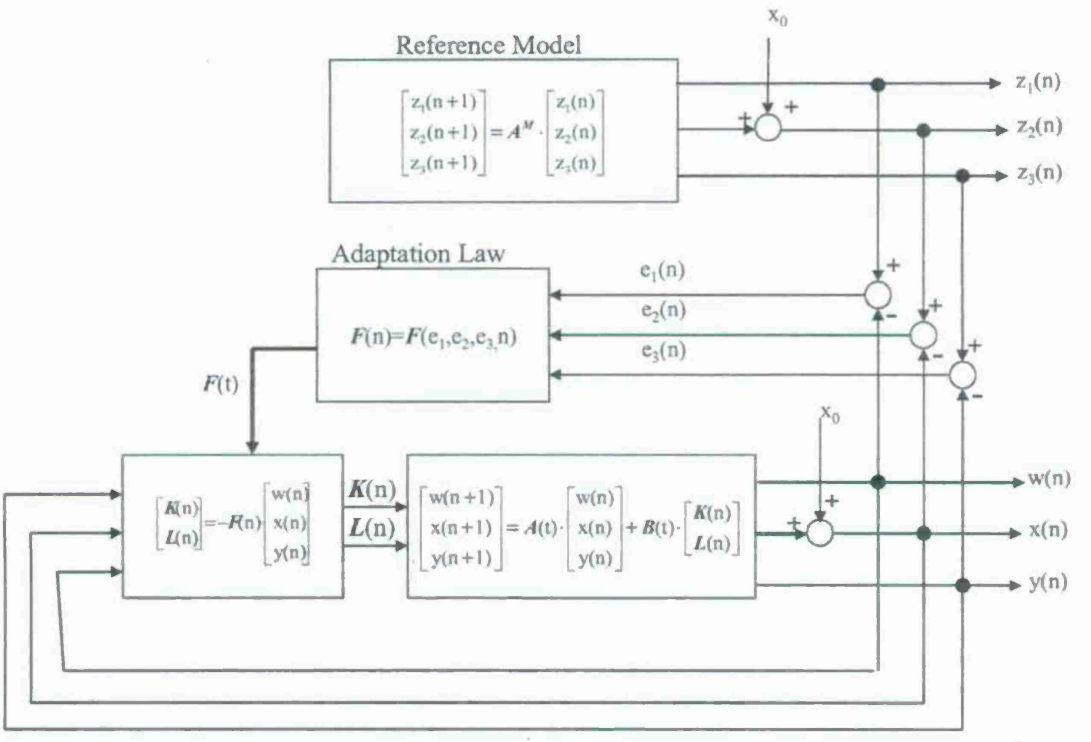


Figure 8: Adaptive Control of the Active Network Response

The error vector,

$$\begin{bmatrix} e_1(n) \\ e_2(n) \\ e_3(n) \end{bmatrix} \tag{20}$$

is calculated as a discrepancy between corresponding states of the reference model and active response and provides the input information for the adaptation block implementing the adaptation law. This law implements one of several techniques described in [50] resulting in *perfect asymptotic adaptation* that implies that all three states of the active immune response, the number of infected hosts, the number of active anti-worm units, and the increment of the network bandwidth will converge to zero regardless of initial conditions, i.e. x_0.

There are several approaches leading to the definition of the control law. The hyperstability and positivity approach [50], developed in model reference adaptive control is based on the linearization of the mathematical model of the network response, design of a state-variable controller with adjustable parameters, and the utilization of adaptation laws to assure the conformance of the parameters of the controller to the immediate operational regime of the inherently nonlinear, time-varying system. The method based on Liapunov stability conditions deals directly with the nonlinear mathematical model of the controlled process and also leads to the synthesis of a model-reference adaptive controller [50]. The model reference controllers have been successfully designed and verified by the application to the nonlinear mathematical model of the network response implemented in MATLAB.

It should be noted that the development and deployment of the control technology described herein constitutes a large-scale task that in terms of the required funding and volume of work goes far beyond the scope of this project. It has been proposed to the AFOSR as the next stage of this research.

5.0 ENHANCED SYSTEM CALL DOMAIN IDS

Major threats, well recognized by government, private institutions and individual users, are stemming primarily from self-proliferating malicious software such as network worms. Network worms perpetrating remote code execution attack, such as buffer overflow, stack overflow, heap overflow and etc, have two vital components, propagation engine (shell code) and an exploit. The shell code being a necessary part of the propagation engine is executed by the vulnerable process just after the exploit vector rerouted the control flow. The shell code creates specific conditions which are utilized by the attacking worm to complete propagation session. Hence, every network worm performing remote code execution attack, utilizes particular type of propagation engine and a corresponding shell code in every attack to replicate into the victim machine.

On the other hand, adversaries usually utilize standard propagation engines along with available exploits of the selected vulnerability. It could be explained by the fact that reverse-engineering of the services, determining vulnerability, developing the exploit and producing specific shell code (propagation engine) requires special experience and knowledge which is possessed by certain small community of people. The biggest part of the worms is written by so-called "script–kiddies" [1], who utilize available exploits with standard shell codes as buffers in the attack packets and implement their own payload what results in the new instances of the worm.

Moreover, a worm family may spawn many strains, so-called versions of the original worms. These strains are created in order to avoid matching to existing binary signatures of known anti-virus databases and to prolong the life-cycle of the worm. While recompiling the original worm code into novel strains, the adversaries change image name, synchronization object name, registry records and employ equivalent code modification techniques. However, according to our experience, versions of one worm family tend to share one or two propagation engines. For instance, W32.Sasser worm has seven known strains, while W32.Padobot worm has 29 strains and in spite of their mutations they utilize only two propagation engines. This is a good demonstration of the fact that the source code of the worms could be easily subjected to change to break known simple signatures, but adversaries hardly change shell coder buffer since it requires to program in specific, base-independent style[1] what involves much more efforts. Even if and an effort is made to change the shell code (propagation engine) to make it binary different, one has to preserve the initial system call pattern to achieve the propagation effect.

As a result, most of the network worms share the same trivial propagation engines and shell codes. To achieve the propagation effect, the shell code must invoke system calls through API functions to utilize operation system (OS) resources. Consequently, shell codes of the worms having the same propagation engine have the same realization in system call domain. Therefore,

---

[1] It requires computing delta offset, searching for entry address of API functions in DLLs and performing variable address alignment.

the task of detecting worms can be narrowed down to recognizing the standard propagation engines utilizing the system call signatures.

Based on highly successful dynamic code analyzer (DCA) concepts [2], the authors developed and evaluated Propagation Engine Detector (PED) system capable of detecting attacks perpetrated by network worms. This system detects the worm shell code activity performed by a process during the attack session. Moreover, PED recognizes the type of propagation engine employed by the worm exploiting the process. The developed PED system utilizes Colored Petri nets (CP-net) to trace in parallel interrelated chains of system calls issued by the monitored process to recognize high level API functions invoked by the process. The detected high level functions in combination are analyzed to determine how the process creates and manipulates operation system objects. Such information is finally processed by CP-net to detect if the process activity exhibits the functionality of the particular type of propagation engine.

5.1. Background and Related Work

System call-based Intrusion Detection Systems (IDS) utilize two main approaches, misuse detection and anomaly detection. Misuse detection or so-called signature-based detection systems employ known traces of system calls to detect malicious activity. This approach ensures high level of accuracy, but fails to detect previously unknown attacks. Anomaly-based detection utilizes models of normal behavior of legitimate and especially privileged processes with respect to invoked system calls. In the detection mode, these systems check consistency between invoked system calls and the profile of normalcy for a given process. Anomaly-based IDS are able to detect unknown (new) attacks, but suffer from high rate of false positives.

Up to now a number of anomaly-based as well as misuse-based IDS have been proposed. The systems could trace merely order of system call execution. The efficiency could be further enhanced by analyzing arguments of system calls.

Anomaly detection using system call without attributes

Forrest at al. [57] proposed to build an n-gram model of normal activity comprising possible sequences of system calls with n elements. To build such a model, they monitor process behavior in both synthetic and real environments. During the detection phase, the IDS evaluate the hamming distance between the current sequence of system calls and the normalcy model. Then, if the distance is greater than a specified threshold, the sequence is attributed to anomaly.

Durante, Pietro and Mancini [58] model the application behavior as a Finite State Machine (FSM), which accepts legitimate system call execution sessions caused by particular user commands. This approach requires comprehensive learning with expert who will trigger all possible commands of the application. If FSM does not accept the sequence of system calls invoked after given command, intrusion alarm will be triggered. This method is not applicable for a process which does not have user interface such as native services being most frequently chosen as targets for attacks.

Stolfo, Eskin and Lee [59] utilize call execution trees to derive a prediction model trough Sparse Markov Transducers. They also suggest using dynamic context-dependent window of contiguous system calls invoked by the process. During the detection phase they check predictive

(conditional) probability of the given subsequence against some threshold, then, if probability is less than threshold, the subsequence is declared anomalous.

Anomaly-based systems which do not explore system call attributes usually show weak performance, since lots of critical information is discarded. For instance, network worm shell code may start command interpreter through "CreateProcess" function with inputs and outputs associated with a socket, what could be determined only by inspecting system call attributes. Moreover, without system call attribute data, it is impossible to relate system calls to some functional chains that is necessary to trace an OS objects manipulation session.

Anomaly detection using system call attributes

Liu and Martin [60] use system calls traces for detecting insider threats when a privileged user tries to perform a malicious activity. They utilize three different feature spaces to build a model of normalcy: n-grams of system call names, histograms of system call ID over fixed window and system calls with attributes. Each system call along with attributes is mapped onto its binary feature space so that dimensions are assigned to distinct values of every parameter. They use minimum hamming distance for n-gram models and system calls attributes models to detect anomalous records.

Tandon and Chan [61] use rule-learning algorithm to derive a model of benign behavior for every process. They take into account all arguments for individual system calls or bag of system calls (contiguous sequence) and build a set of specific rules. Feature vector for a single system call consists of such qualitative elements as: ID of system call, arguments, returned value and error status. Feature vector for a bag of system calls is composed of concatenated feature vectors of corresponding system calls from the bag. During the detection phase, feature vectors, inconsistent with the rules, are considered as anomalous with some degree of certainty.

Xu at al. [62], in contrast, analyze only critical system calls, which are vital for gaining access or control to the privileged target system. Their method also generates a pre-defined set of rules constituting the profile of normal behavior. Authors group system calls and assign level of danger to every system call. Nevertheless, they do not cohere system calls in functional chains what results in subjective information failing to perform reliable detection.

Kruegel at al. [63] suggests using arguments of file management system calls which represent file names to be manipulated or accessed. Authors depict four models of normal strings of file name arguments: string length, character distribution, string structure and token structure. Having these four levels of model abstraction, the system based on this approach is able to detect malicious activity in terms of abnormal argument strings. Since the method is focused only on file manipulation it would not be able to detect "Executable Download and Execute" shell code in the case when the name of the worm image is consistent with the string models.

It is our observation that the limitation of n-gram and frequency-based models lies in specifics of input data. These methods disregard functional relation between system calls and trace only contiguous system calls which may not be related by any functionality. This seems to be the main reason of high false positive rate reported by the authors. The problem with attribute models is that comprehensive learning may require large amount of learning space to contain all

distinct records of models for all system calls for all processes. In addition, these models do not classify and group system calls to functional blocks, and the testing may result in high look-up time. In contrast, the success of the earlier mentioned DCA approach could be credited to the full utilization of attributes of system calls enabling the authors to reconstruct the "gene of self-replication" on a block-by-block basis [2].

Misuse detection

Bernanshi, Gabrielli, Mancini [64] classify system calls with respect to the feasibility of utilizing individual system calls in compromising OS security and integrity. The IDS also contains the database of access control rules defined in the system call domain. Thus, the efficiency of the system depends on completeness of the rule base for particular process. For instance, the IDS will block any attempt to run an executable which is never executed by the mediated process. While the authors mention the necessity of chaining system calls in order to reveal dangerous calls combinations, they do not provide any mechanism for assembling system calls into functional blocks. Hence, the system detects malicious activity only based on single system call misuse, what makes the security decision subjective and less reliable.

Kahg and Fuller [65] employ system call frequency distribution over fixed length window as in the input feature space and apply machine learning algorithms to make classification between malicious and normal system call traces. Again, the authors do not take into account high level functionality and system call semantics.

Sekar and Bowen [66] also developed high-level specification language to profile system calls usage with arguments for every process. Sequences of system calls which conflict with the rules of the language are treated as anomalous. While the authors trace sequences of calls with respect to attributes, they do not deduce explicitly OS object manipulation to reveal semantics of the system call chains what seems to be necessary to recognize a propagation engine.

In summary, some authors propose to retrieve quantative information from system call issuance data; others – to pre-classify system calls to explore categorical information. While such information does not provide enough knowledge to detect malicious activity with high confidence, there were a few attempts [64, 66] to deduce semantic information on the level of primitive functional blocks. However, recognizing merely primitive functional blocks would not provide complete picture of the process behavior.

We believe that without tracing the entire functionally of the shell code by restoring the complete procedure of OS object manipulation confident detection decisions cannot be reached. To address the shortcomings of the referenced approaches, we propose to reconstruct the entire procedure of OS object manipulation revealing particular high level operations and finally recognizing semantically expressed high level activity as a shell code algorithm. The proposed PED system implicitly conducts all the recognition steps while simulating the dedicated Colored Petri net.

5.2. Analysis of Propagation Engine Utilization

The largest library of exploits hosted by Metasploit Project [67] provides 18 possible shell codes for Windows OS, which could be attached to the exploit vector and potentially employed to

compromise security of a susceptible host. Shell codes 1, 2, 3, 6 and 7 could be effectively utilized as a part of the propagation engine to achieve a worm proliferation into the target host. However, according to our observations, adversaries inherently employ limited set of particular types of propagation engines in the network worms.

Table 4. Standard Shell Codes Available in Metasploit Project

1	Bind Shell
2	Reverse Shell
3	Bind DLL Inject
4	Bind Meterpreter DLL Inject
5	Bind VNC Server DLL Inject
6	Executable Download and Execute
7	Execute Command
8	Execute net user /ADD
9	PassiveX ActiveX Inject Meterpreter Payload
10	PassiveX ActiveX Inject VNC Server Payload
11	PassiveX ActiveX Injection Payload
12	Recv Tag Findsock Meterpreter
13	Recv Tag Findsock Shell
14	Recv Tag Findsock VNC Inject
15	Reverse DLL Inject
16	Reverse Meterpreter DLL Inject
17	Reverse Ordinal VNC Server Inject
18	Reverse VNC Server Inject

To estimate the tendency of propagation engine utilization, we investigated 25 recent network worm families including: Sasser, Welchia, Blaster, Slammer and Mytob. Figure 9 depicts the propagation engine distribution among the studied worms. Worm propagation engines were determined based on Symantec virus data base as well as reverse engineering particular strains of the worms. It could be observed that more than 60% of the worms employ "Bind shell" engine, while "Reverse shell" and "Executable Download and Execute" (ED&E) propagation engines are shared by 30% of the worms. Finally, less than 10% of the worms utilize other types of the engines such as thread injection, remote command execution and etc.

The first three engines (Figure 9) are dominantly employed in the worms due to the simplicity of their utilization in the propagation session.

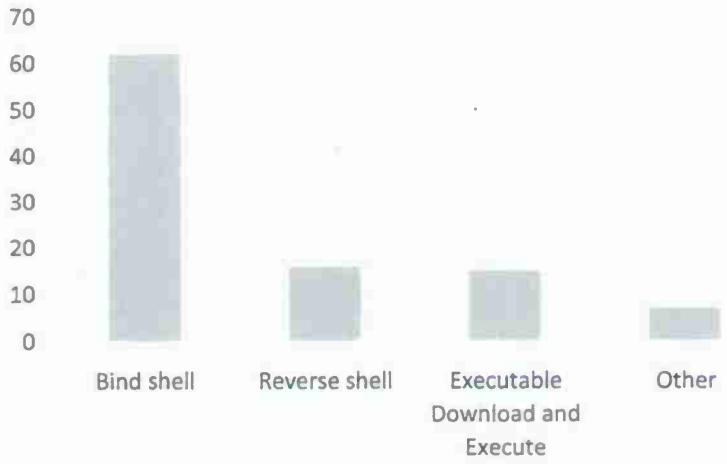


Figure 9: Propagation Engine Utilization

Table 5 explains the high level operation of these engines. The first row of the table summarizes the functionality of the shell code of the propagation engines. The second row reviews the activity of the rest of the propagation engine which is not performed in the context of the exploited process, but by the legitimate means of OS.

Table 5. Propagation Engine Operation

	Bind shell	Reverse Shell	Executable download and Execute (ED&E)
Shell code (executed by the exploited process)	Open a port Accept connection Create a command interpreter process that will listen on the port for incoming commands. (This will allow an attacker to issue remote commands on an infected computer).	Open a port Connect to the attacker Create a command interpreter process that will listen on the port for incoming commands. (This will allow an attacker to issue remote commands on an infected computer).	Connect to the server in the attacker host. Download the worm image Execute the worm image.
The rest of the engine (performed legitimately by OS)	Transmit commands to the victim host and make it to download worm image and execute it. For instance through TFTP.	Transmit commands to the victim host and make it to download worm image and execute it. For instance through TFTP.	

The "Reverse Shell" engine is very similar to the "Bind Shell". However, in order to avoid inbound firewall, the shell code makes victim to connect to the attacker, instead of waiting for connection from the attacker.

The ED&E engine performs the entire propagation in the shell code without post-activity as it is performed in first two engines. It simply creates a socket, establishes a connection to the attacker and retrieves a copy of the worm through the established channel. The shell code usually uses facility of high level protocols such as HTTP to download a worm, but sometimes it downloads the worm directly through the channel using merely TCP.

The above considerations indicate that different worms form different families tend to share the same propagation engines. And the number of totally different types of propagation engine is limited. At the first stage of the propagation session, the worm shell code is executed by the target, the vulnerable process. To achieve propagation effect, the shell code has to utilize system resources trough utilizing API functions. As a result, each type of shell code would have its own system call execution pattern. Hence, one can detect and recognize particular system call signatures of the propagation engines. Therefore, detecting worm attacks can be narrowed down to recognizing the propagation engines based on the system calls signature.

5.3. Recognition of the Propagation Engine

As it was established above, in the first stage of propagation, the shell code invokes high level API functions. For instance, consider the operation of the shell code of the "Bind Shell" engine on the subsystem (API) level. It was noted above that the "Bind Shell" engine has quite primitive shell code to be executed by compromised process and this code invokes several high level APIs. One of the realizations of such a shell code for Windows OS is presented in Table 6. It could be seen that socket object is created using Socket API than it is put on listening state through Listen function and after accepting a connection the command interpreter ("cmd.exe") is started by CreateFile function with input and output set to the socket handle.

Table 6. Bind Shell Engine High Level Implementation

	API (function)	Parameters	Description
1	s=socket(…)	_out s	Opens a socket
2	bind(s)	_in *socket*=s	Bind the socket
3	listen(s)	_in *socket*=s	Put the socket to listen state
4	s1=accept(s)	_in *socket*=s _out s1	Accept a connection to the socket
5	CreateProcess("cmd.exe",…)	_in *pszImageName*="cmd.exe" _in *STARTUPINFO.hStdInput*=s1 _in *STARTUPINFO.hStdOutput*=s1	Start command interpreter with standard output and input being tied to the connected socket

A propagation engine may have several realizations with respect to high level functions. However, according to our experience, such high level variations tend to preserve the same implementation with respect to system calls. For instance, bind shell engine may be realized

through anonymous pipes as well as through API's listed in Table 6. However, both high level realizations get translated into the same sequence of system calls. Hence, for the sake of reliability, propagation engines should be recognized at the system call level.

In order to recognize the shell code type, one has to model shell code activity in system call domain. System calls by themselves do not provide a complete picture of the process activity, however system calls create and manipulate objects by means of handles and other system descriptors. Hence, we are more concerned of what shell code does with OS objects. For instance in the realization presented in Table 6, the "Bind Shell" engine creates an OS object named socket and puts it in the listening state to accept a connection. Thus, in case of "Bind Shell" engine, we have to trace socket manipulation in order to detect the malicious functionality. To track object manipulation we have to relate system calls by object handles. In other words, the model has to follow inheritance of the object handles used by system calls and reveal chains of system calls. However, the shell code may have several chains not related to each other until some specific moment. Hence, the model must have a memory and an ability to trace system call chains in parallel. Therefore, it is required to provide parallelism, memory and inheritance, what could be addressed by utilizing Petri Nets [68].

Since system calls are related by the handle value, Colored Petri Net (CP-net) must be utilized to formally describe the manifestation of a propagation engine in the system call domain. CP-nets are able to model particular activity in terms of actions (transitions) and states (places) at any desired level of abstraction thus providing the necessary generalization. Such a general model would be able to recognize the type of propagation engine in spite of possible code modifications or platform variances. Moreover, due to the parallel structure of the Petri nets the model realization would be compact causing low computational overhead.

The CP-net could formally be presented as a tuple [68]:

$$CPN = (C, P, T, A, N, F, G, E, I) \tag{21}$$

where: C – color set, P – set of places, T – set of transitions, A – set of arcs and etc.

The *color* sets constitute OS object handles employed by system calls and other system descriptors such as file names.

To address the specifics of our problem, we had to extend the original formalism of the CP-net in the way that *set of places* consists of tree disjoint dedicated subsets:

$$P = S \cup B \cup R \tag{22}$$

where S – set of system calls places, B – set of functional blocks of system calls, R – set of recognition nodes. Thus, each place of the set S corresponds to particular system call and its tokens are defined as ordered pair of certain attributes of the executed system call. Places of the set B contain tokens representing successful execution of corresponding functional block. Recognition places from R represent successful recognition of the propagation engine.

Another extension we introduced into the classical CP-net is the inclusion of *inlet and outlet transitions*. The inlet transitions correspond to the system call execution which results in firing a

new token to the corresponding place. The outlet transitions represent handle elimination, for instance through NtClose system call, which results on destroying token from the corresponding place usually belonging to the set S. Ordinary, transitions assemble system calls into chains (functional blocks) represented by places from the set B. Hence, each token represent an instance of execution of particular chain of system calls interconnected by object handles, pointers, etc.

In the context of one process different objects cannot share the same handle value. Hence the CP-net is free of conflicts what significantly simplifies implementation since we do not have to carry out conflict resolution policy.

Figure 10 shows a reduced version CP-net for the "Bind shell" propagation engine. The set of places $P = S \cup B \cup R$ of the full version of CP-net is defined in the following way:

$$S = \left\{ \begin{array}{c} \text{NtCreateProcess|NtCreateProcessEx,} \\ \text{NtCreateFile|NtOpenFile} \\ \text{NtWriteFile,NtWrtieVirtualMemory,} \\ \text{NtDeviceIOControl,NtClose} \end{array} \right\} \tag{23}$$

$$B = \left\{ \begin{array}{c} \text{"FileSection","FileSectionProcess"} \\ \text{"SocketOpen","ShellOpen"} \end{array} \right\} \tag{24}$$

$$R = \{\text{"}BindShell\text{"}\} \tag{25}$$

The network has a recognition node which represents an instance of "Bind Shell". Due to space limitations, the Petri network is depicted without token outlet nodes which reflect object elimination through *NtClose* system call. Moreover, the network misses several alternative (undocumented) system calls which act the same as the original ones. For instance *NtCreateProcessEx* is functionally identical to *NtCreateProcess*, but has different entry point address. The prototype IDS employs full version of the Petri net with all necessary components included.

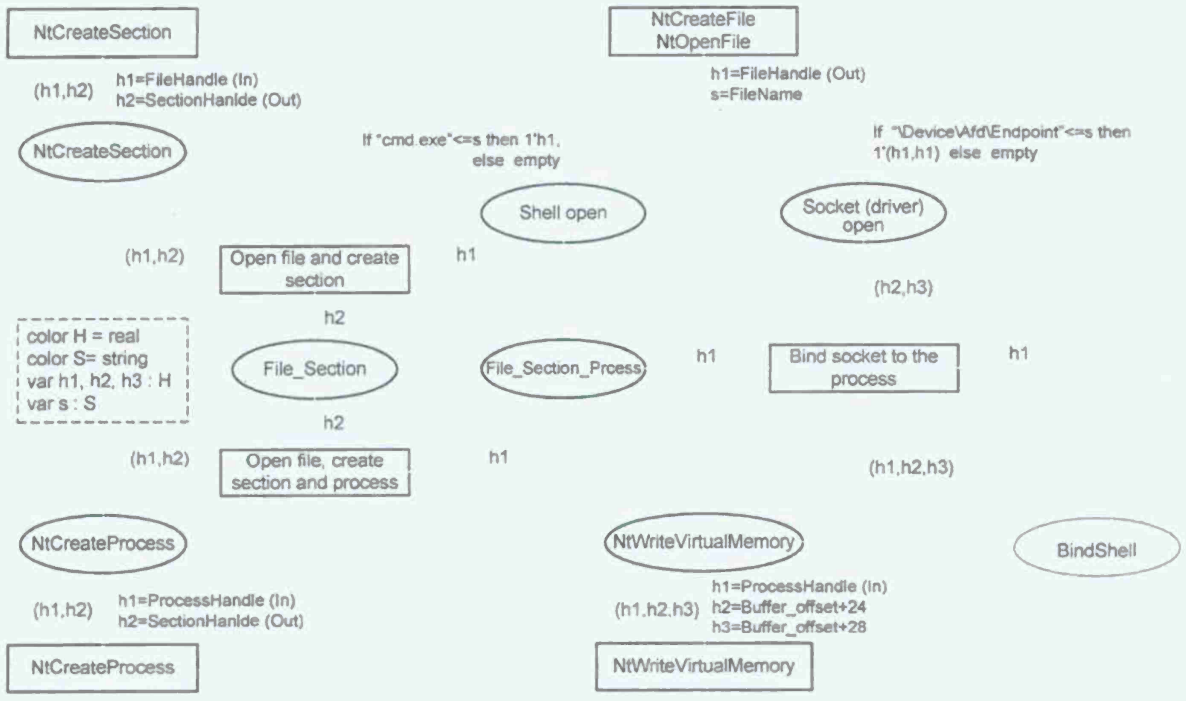


Figure 10: CP-net for Bind Shell Propagation Engine

The CP-net in Figure 10 has two color sets (types). The former type (H) is a handle and its variables represent object handles. The later color set (S) is a string which describes names of the files. Color and variable declarations listed in Figure 10 are written using syntax of CPN markup language [68]. Also, the network has four inlet transitions: NtCreateSection, NtCreateFile, NtCreateProcess and NtWriteVirtualMemory. These transitions are enabled at the moment of successful execution of the corresponding system calls. The outgoing arcs of the system call transitions are provided with inscriptions which define token structure and variable initialization. For instance, when *NtCreateProcess* system call is invoked, the corresponding transition is enabled and fires a token which constitutes an ordered pair of two handles: handle of the process (input parameter) and the section handle (output parameter). There are also ordinary transitions such as "Open file and create section", which are enabled with particular events such as execution of the related chain of system calls. We used complex arc expressions to minimize structure of the network. For instance, expression of the ingoing arc to "Shell open" place checks if the file name contains the "cmd.exe" string, and if it does the network fires file handle as a token to the place, otherwise it does not put anything.

The transition *NtCreateFile* is enabled if the corresponding system call has been executed. One can see that the token can be fired to one of the two places depending on the result of outgoing arc expressions with *FileName* parameter. If the file name contains "\Device\Afd\Endpoint" string, then it means that the OS opens an instance of the AFD.sys driver which abstractly emulates a socket object. In fact, *Socket* function from *WS2_32.dll* invokes *CreateFile* function to open the driver instance and the handle of the driver will be ultimately employed as a handle of the socket object. Hence, in the Petri net a token pair of the handle is added to the "Socket open" place. This means that socket is created and is ready to be bound to a process.

The system calls *NtCreateSection*, *NtCreateFile* with "cmd.exe" and *NtCreateProcess* simply perform necessary steps for starting "cmd.exe" process. We should point out, that only properly related system calls result in transition fire. Hence, transition "Open file and create section" is enabled if *NtCreateSection* system call have file handle being equal to the handle of the opened "cmd.exe" file. Therefore, this transition fires section handle as a handle token to "File_Section" place only if ingoing tokens match each other (values of h1 variables are equal in both tokens). Place "File_Section_Process" corresponds to the state at which the address space of the process is allocated and Windows is ready to initiate the process. System call *NtWriteVirtualMemmory* maps *CreateProcess* parameters to the process address space. The transition "Bind socket to the process" is enabled if the input and output handles of the process equal to the handle of the socket what indicates binding the opened socket to the process.

We intentionally omitted some minor system calls as transitions in the network. In particular, in the real network utilized in the prototype IDS, there is NtDeviceIoControlFile transition (system call) located after "Socket (driver) open place". This system call sends commands to the driver ordering it to put the socket on listen or accept state. Moreover, we did not include *NtCreateThread* and *NtResumeThread* transitions (system calls) which constitute final steps in the process creation and running.

The CP-net depicted in Figure 10 is not merely low level map of high level implementation of the "Bind Shell" engine. Since, according to our experiments, different realizations and modifications of the original engine are detected by the same network, the Petri Net in Figure 10 is a general signature of the particular type of the propagation engine.

Furthermore, we designed CP-nets for the shell codes of such propagation engines as ED&E and "Reverse shell". Parallel structure of Petri Nets allowed us to merge several networks of different engines into one general, multi-engine network. The simplified version of the general network is depicted in Figure 11. This network is able to recognize "Bind Shell", "Reverse Shell" as well as ED&E propagation engines.

The multi-engine network has three recognition places which represent successful detection of the corresponding propagation engines. The part of the network which recognizes "Bind Shell" engine is structurally similar to the network depicted in Figure 10. However, file name (variable *s*) is delivered to the place "Executable started" what allows to recognize the bind shell engine if file name is "cmd.exe", or the second engine if the executed file was prior downloaded.

The network also contains two major states "Executable started" and "File downloaded". "Executable started" corresponds to successful execution of *CreateProcess* API (kernel32.dll). "File download" could be achieved through the sequence of high level APIs (wininet.dll - InternetOpenURL, InternetReadFile, CreateFile, WriteFile), or through subsystem level implementation (kernel32.dll – Socket, Connect, Send, Recv, CreateFile, WriteFile). However, both high level realizations of "File download" functionality have the common implementation in system call level which involves *NtDeviceIOControlFile and NtWriteFile* system calls.

The general network is more compact due to the points of integration which are the parts related to *NtOpenFile* and "Executable started" place. The node "Executable started" is shared as an input by the recognition places of the corresponding engines. Hence, the entire substructure of

the network (upper-left), which is involved in token delivery to the "Executable started" place, is shared and employed in recognizing of three separate engines. Such integration on the structural level significantly reduces size of the overall network.

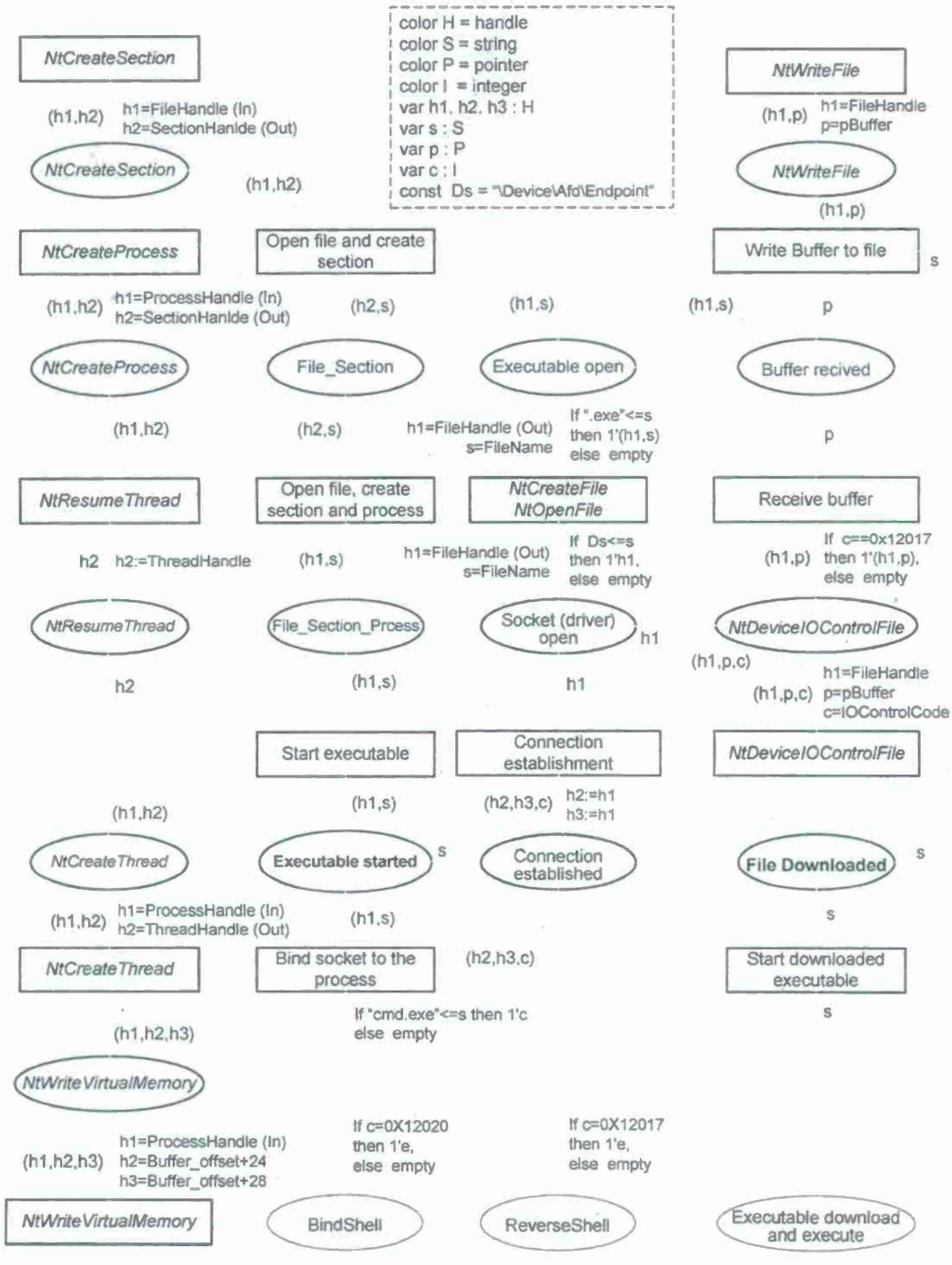


Figure 11: General "Multi-Engine" CP-net

5.4. Experimental Evaluation

In this section we evaluate the performance of the proposed approach. We designed a general CP-net capable of recognizing three propagation engines: ED&E, "Bind Shell" and "Reverse Shell". Moreover, we implemented prototype of Propagation Engine Detector (PED) based on the designed CP-net. PED monitors system calls of the processes and recognizes propagation activity as a shell code.

Experimental setup

The experiments were conducted in the virtual network testbed at Binghamton University [69]. The testbed, being scalable up to 2000 nodes, was configured for virtual network comprising 200 victim hosts and one attacker host. The victim hosts were represented by virtual machines with vulnerable versions of Windows OS including the prototype PED. The attack was performed by a worm from the attacker host against each victim hosts resulting in worm propagation into the victim virtual machines.

We experimented with 10 notable network worms presented in Table 7. Every worm has been reverse-engineered deeply up to the point of revealing the exploit vector and shell code buffers supposed to be sent as payload in the attack packets. Some of the worms (Francette, Welchia) have shell code buffer encrypted and decrypt it just before sending. Hence, in order to extract exploit and shell code buffers we had to utilize run-time debugger and TCP dump software to record and process attacking packets payload. While reverse-engineering is not a trivial task, we have spent even more efforts to extract exploit and shell code of the propagation engines of the worms.

Having exploit and shell code buffers, we were able to incorporate them into a custom, educational worm which was specifically implemented for the experiments. The custom worm is completely observable and controllable and allows performing user specified attack scenarios. Every realization of the custom worm with various shell codes and exploits share the single payload which simply reports to the control server every propagation step: arrival, deployment, starting of the attack session and proliferation status. The custom worm also provides necessary functionalities for completing propagation session for different engines. Such functionalities include: TFTP/FTP server for "BindShell" engine and TCP server for uploading the image in case of utilization ED&E engine.

Utilization of the custom worm with different shell codes of the originally tested worms provided test unification and dependability of the results without sacrificing generality of the experiments. Moreover, many worms have imperfections in certain modules utilized in propagation session, for instance W32.Welchia.A worm has bugs in TFTP server what may result in unsuccessful replication. However, the custom worm has more reliable modules and its propagation rate could be much higher.

Experimental Results

We performed two sessions of experiments. In the first session, we deployed attacks from the server host to victim machines utilizing the custom worm equipped with propagation engines of the worms subjected to the test (Table 7). During attacks the target hosts were in idle mode. In

other words we did not perform or emulate user activity on the victim hosts during the attack session. The main goal of these experiments was to estimate false negatives and we did not observe any.

Propagation engine of every worm in the Table 7 was employed in 200 consequent attacks perpetrated from the attacker server against randomly selected victim hosts. During these attacks our system monitored corresponding vulnerable processes and correctly detected and recognized propagation engines for every attack session indicating no false negatives and no false positives.

It could be seen that the original worms utilize different exploits and some of them attack different windows versions. However, we used the same version of PED in both windows versions. The last column depicts propagation engine and a protocol used for retrieving the worm to the victim machine. We should point out that W32.Ibero worm and W32.Shelp worms utilize very different realizations of the executable download and execute engines. W32.Ibero directly retrieves a worm copy through the open TCP channel. And W32.Shelp uses HTTP commands to download worm image from the dedicated web site. However, in spite of realization differences in engines, PED successfully recognized the propagation engine type what indicates high reliability of the proposed approach.

Table 7. Network Worms Being Tested

Worm name (aliases)	Vulnerability (MS code)	Target system	Propagation engine (upload protocol)
W32.Welchia.A	DCOM RPC (MS03-026)	Win XP Sp1	Reverse shell (TFTP)
W32.Sasser.C	LSASS (MS04-011)	Win 2000 Sp4, Win XP Sp1	Bindshell (FTP)
W32.Zotob.F (Bozori.A)	Plug and Play (MS05-036)	Win 2000 Sp4	Bindshell (TFTP)
W32.Iberio (Hiberium.B)	Plug and Play (MS05-036)	Win 2000 Sp4	Executable download and Execute (direct download through TCP channel)
W32.Raleka	DCOM RPC (MS03-026)	Win XP Sp1	Bindshell (ECHO, direct injection)
W32/Alasrou-A (Small.D)	LSASS (MS04-011)	Win2000 Sp4, Win XP Sp1	Bindshell (TFTP)
W32.Kassbot(W32.Nanspy)	DCOM RPC (MS03-026)	Win XP Sp1	Bindshell (TFTP)
W32.Shelp	LSASS (MS04-011)	Win 2000 Sp4, Win XP Sp1	Executable download and Execute (HTTP from dedicated site)
W32.Blaster (Lovesan)	DCOM RPC (MS03-026)	WinXP Sp1	Bindshell (TFTP)
W32.Francette	DCOM RPC (MS03-026)	Win XP Sp1	Bindshell (TFTP)

The second experimental session was performed on particular, attack free hosts to determine false positives. The PED system monitored regular legitimate processes maintaining most windows services such as: lsass, svchost (RPC DCOM), winlogon, csrss and etc. PED also observed processes representing several applications such as: word processors, picture editors, file managers and internet browsers. On these virtual machines we browsed internet, manipulated files, managed various windows components and performed other activity trivial for an advanced user. None of the monitored processes caused false positive, since they did not exhibit any behavior being attributed to the shell code activity.

The outcome of the experiments described above indicates zero false positive and zero false negative for limited set of legitimate processes and ten worms (Table 7) tested during limited time period. While authors believe that the proposed approach is able to detect most of the existing worms and any feature worms with standard propagation engines, such high detection rate is not guaranteed to be consistent for the feature malware attacks. The authors do realize that no detection technique is perfect and it is expected that some sophisticated adversary may create a worm with totally new propagation engine which is not yet reflected in the CP-net. However, a new propagation engine must have certain pattern in system call domain and may be easily incorporated into the Petri Net of PED ensuring feature detections of the worms based on the new engine.

6.0 CONTINUOUS STATUS ASSESSMENT OF A COMPUTER NETWORK

Implementation of an automatic network defense system described in Section 4 requires continuous estimation of the number of infected hosts and the activity level of the anti-worm. It could be seen that this task cannot be accomplished without a network scanning activity conducted in a worm-like fashion, however this would have a negative impact on network bandwidth. Performing this task with minimum scanning presents an important problem that was not addressed in known literature. Existing publications have addressed only the scanning issue in light of passive network assessment being a part of network intrusion detection systems [69]. Passive network assessment is performed in order to detect an actual worm outbreak based on suspicious activity by listening to unused IP space, detecting malicious scanning and evaluating the dynamics of worm propagation. This may not be applicable for accurately estimating the number if infected hosts in the presence of both worm and anti-worm activity.

The scanning approach presented herein has different goals, and could be addressed as active scanning. First, it is performed after the worm has been detected. Second, we control the level of scanning in order to minimize the impact on bandwidth in the presence of both worm and anti-worm activity. Third, we have a means for managing estimation error.

In order to obtain the most reliable network status information (such as the number of infected hosts), one has to scan the entire network with some periodicity, consistent with the dynamics of the worm propagation process. For large networks such as wide-area or corporate class local networks, scanning all hosts from several dedicated nodes may consume a significant portion of the available bandwidth and negatively affect network operations. For middle and small sized local networks a single scan of every host will not consume much bandwidth. Nevertheless, due to modern worm's fast propagation rates, scanning must be performed at a very high rate, causing the same detrimental effect on the bandwidth. Hence, scanning all of the individual hosts on a network in a purely worm-like fashion is clearly unacceptable. However, one can take advantage of a selective scanning approach, which implies scanning some randomly chosen group (a small sample) of computers that represents the entire network. Such an approach, commonly utilized in quality control, allows one to estimate the number of infected computers in the network using statistical inference techniques without significantly impacting network bandwidth.

6.1. Network Status Assessment

To minimize session time as well as bandwidth consumption by scanning activities, one has to minimize the number of hosts that are scanned during each session. However, decreasing the sample size will increase the estimation error. Hence, we have to investigate a trade-off between estimation error and scanning sample size.

The estimation error can be defined in relative as well as absolute terms. When the estimate of some variable has a large value, there is usually more concern about relative error, while for small values of the estimate, the absolute error is more important. As a result, in this context, we can formulate the following estimation problem: define the minimum number of scanned hosts during one session, which ensures relative error or absolute error to fall below some prescribed threshold. Formally such a problem could be presented in the following way:

$$l = \min\left\{ n \in \mathbb{N} \,\middle|\, \left(|\hat{\theta}(n) - \theta| / \theta \le \delta \right) \text{ or } \left(|\hat{\theta}(n) - \theta| \le A \right) \right\} \tag{26}$$

where l is the sample size (number of scanned hosts), θ is the number of infected computers in the network, $\hat{\theta}$ is the estimate of the number of infected computers based on the sample with size n, δ is the specified relative error level and A is the specified absolute error level.

Then the estimate in successive sessions is computed for samples with size l. Before solving the stated problem, one must also consider the underlying statistical distribution to be assumed for the estimates.

6.2. Justification of the Underlying Distribution

The problem of estimating the number of infected computers in the network is technically similar to the problem of estimating the number of defective items in a manufacturing batch process. For a known population size and a fixed sample size, such a problem is usually solved by the method of moments estimation under the assumption of an underlying geometric distribution. However, in reality we do not know the exact number of vulnerable hosts (population size) since computers may be arbitrarily connected and disconnected from the network. Obviously, this will cause an oscillation in the number of reachable (vulnerable) hosts, and as a result, a geometric distribution is not applicable.

On the other hand, if the number of susceptible hosts is sufficiently large, we do not have to know exact number of vulnerable computers to apply the binomial distribution. The binomial distribution has the following formula (mass function):

$$P(m) = \binom{n}{m} p^m (1-p)^{n-m}. \tag{27}$$

where p is the binomial proportion parameter.

In the context of this problem $P(m)$ is the probability that in the sample of n scanned hosts, where m of them are infected. One can assume that the number of the infected computers in the fixed size sample is distributed according to the binomial distribution with an expected value that is equal to the proportion of the infected computers in the entire network. However, to be able to use statistical inference techniques to estimate the expected value, we have to ensure that the sample is identically and independently distributed. In other words, the source must be stationary during the sampling session. For these reasons we impose the following technical requirements on the system:

Requirement 1: Queried hosts are chosen uniformly and randomly.

Requirement 2: During the sampling process the selected number of vulnerable hosts and the number of infected machines do not change.

The first requirement provides independence and can be achieved using a random generator. The second requirement enforces the identity property and can be satisfied only if computers are scanned pseudo simultaneously.

6.3. Solution of the Estimation Problem

The portion of the infected hosts in the network can be obtained using a confidence interval, which would contain a true value with some specified probability. Note that for the $(1-\alpha)$ confidence interval $\theta \in [c(n,\alpha)-r(n,\alpha),\ c(n,\alpha)+r(n,\alpha)]$ is based on an n sized simple, the estimate being the center of the interval $c(n,\alpha)$ and would have an absolute error with respect to the true value of the parameter of not more than the radius of the interval $r(n,\alpha)$, with a probability equal to its confidence level. Therefore, we can formalize the estimation problem (26) in terms of confidence intervals in the following way:

$$l = \min\left\{ n \in \mathbb{N} \,\middle|\, \left(r(n,\alpha) \le c(n,\alpha)\cdot\delta\right) \text{ or } \left(r(n,\alpha) \le A\right) \right\} \quad (28)$$

Assuming the binominal distribution (27), we can define a confidence interval for a binominal proportion parameter as being an interval estimate of the share of infected hosts. Therefore, we can solve problem (28) with respect to the proportions of infected hosts in the network. However, to be able to solve the problem analytically, we must utilize the closed-form expression of the confidence interval.

The exact confidence interval for a binomial proportion cannot be obtained in a closed-form by inverting the acceptance region. Hence, there are several approximations of the confidence interval based on the Central-Limit Theorem. Nevertheless, the approximations in analytical form do not guarantee a uniform coverage probability for the entire range of the estimated parameter. In contrast, the Clopper-Pearson exact interval, computed using numerical methods, provides an assigned coverage probability with any specified precision.

Therefore, taking into account these considerations, we propose to use an approximate confidence interval to compute the initial approximation (first guess) of l in (28) and then iterate using an exact interval to achieve the best solution.

The Clopper-Pearson (exact) interval for binomial proportions can be represented by the following set:

$$\{\theta : P[X \le k] \ge \alpha/2\} \cap \{\theta : P[X \ge k] \ge \alpha/2\}, \quad (29)$$

where θ is the binomial proportion (estimated parameter), X is the binomial random number, k is the number of infected computers in the sample, n is the sample size, and $1-\alpha$ is the coverage probability (confidence level).

For the first guess, we can utilize available approximate confidence intervals. The most widely recommended approximate interval is Wilson interval [70], which has the following analytical form:

$$c(\hat{\theta},n,\alpha)=\frac{\hat{\theta}+\frac{1}{2n}z^2_{1-\alpha/2}}{1+\frac{1}{n}z^2_{1-\alpha/2}};\ r(\hat{\theta},n,\alpha)=\frac{z_{1-\alpha/2}\sqrt{\frac{1}{n}\left[\hat{\theta}(1-\hat{\theta})+\frac{1}{4n}z^2_{1-\alpha/2}\right]}}{1+\frac{1}{n}z^2_{1-\alpha/2}} \qquad (30)\ 4$$

where

$c(\hat{\theta},n,\alpha)$ is the center the obtained interval,

$r(\hat{\theta},n,\alpha)$ is the interval radius,

$\hat{\theta}$ is the maximum likelihood estimate of the binomial proportion (sample mean).

For a known $\hat{\theta}$ and specified $(1-\alpha)$ confidence level, we can substitute terms from (30) into formula (29) and solve for l. One can easily verify that the solution can be reduced to simple quadratic equations and solved in closed form.

The minimum sample size brings the estimation error to the specified level. Thus the problem presented by Equation 3 can be reduced to maintaining the assigned relative error or absolute error limits. This requires the use of a feedback control system with swappable controlled outputs. We used proportional-integral (PI) controller in the control loop which is schematically presented in Figure 12. It could be seen that the number of scanned computers plays the role of the control effort and the controlled variable represents the relative or absolute error.

In Figure 12, the PI controller computes sample size and applies it to the network scanner which estimates the number of infected hosts and computes estimation errors. If the deviation of the relative error is less than normalized deviation of the absolute error, then the relative error will be controlled and vise-versa. Controller gains are specified for each of the two controlled errors. In order to prevent undesired frequent controlled output switching, we introduced a limitation on the minimum number of iterations (scanning sessions) before a toggle is allowed. Such an approach minimizes the value of the applied input (the sample size).

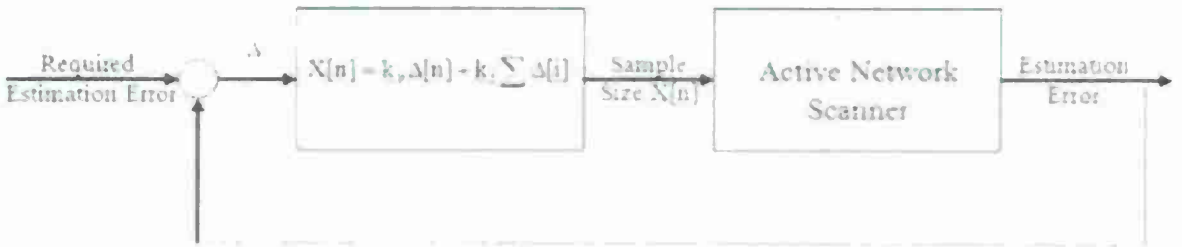


Figure 12: Control Sytem with PI Controller

The procedure implementing the system is presented in Figure 13 in more detail. The following notations were assumed: δ is the relative error set, A is absolute error set, k is the number of infected computers in the current sample, $1-\alpha$ is the coverage probability, X_n is a binominal random variable for n trials (scanned hosts), K_I^{abs}, K_p^{abs} are the assigned integral and proportional

gains for absolute error accordingly, K_i^{rel}, K_p^{rel} are the assigned integral and proportional gains for relative error accordingly, and m is the minimum number of scanning sessions allowed before switching can occur. As the first step, the initial approximation of the minimum size is computed for some initial sample, specified relative error set and absolute error set. Then, in the step 2, n hosts are scanned to generate a new sample. Next, a confidence interval for the scanned sample (step 3) is determined. In step 4, deviation of relative error ε^* as well as normalized deviation of the absolute error $\hat{\varepsilon}$ is computed[2]. If relative error deviation was larger than absolute error during at least m iterations (scanning sessions), then the controlled output will be assigned to the relative error and the PI controller will switch to the corresponding gains. Similarly, the controller will be switched from relative to absolute error mode if the opposite situation is encountered. This control procedure will maintain relative error or absolute error based on the specified level with a minimum number of scanned computers in the sessions.

[2]The absolute error discrepancy is used in normalized form in order to be able to compare to relative error discrepancy.

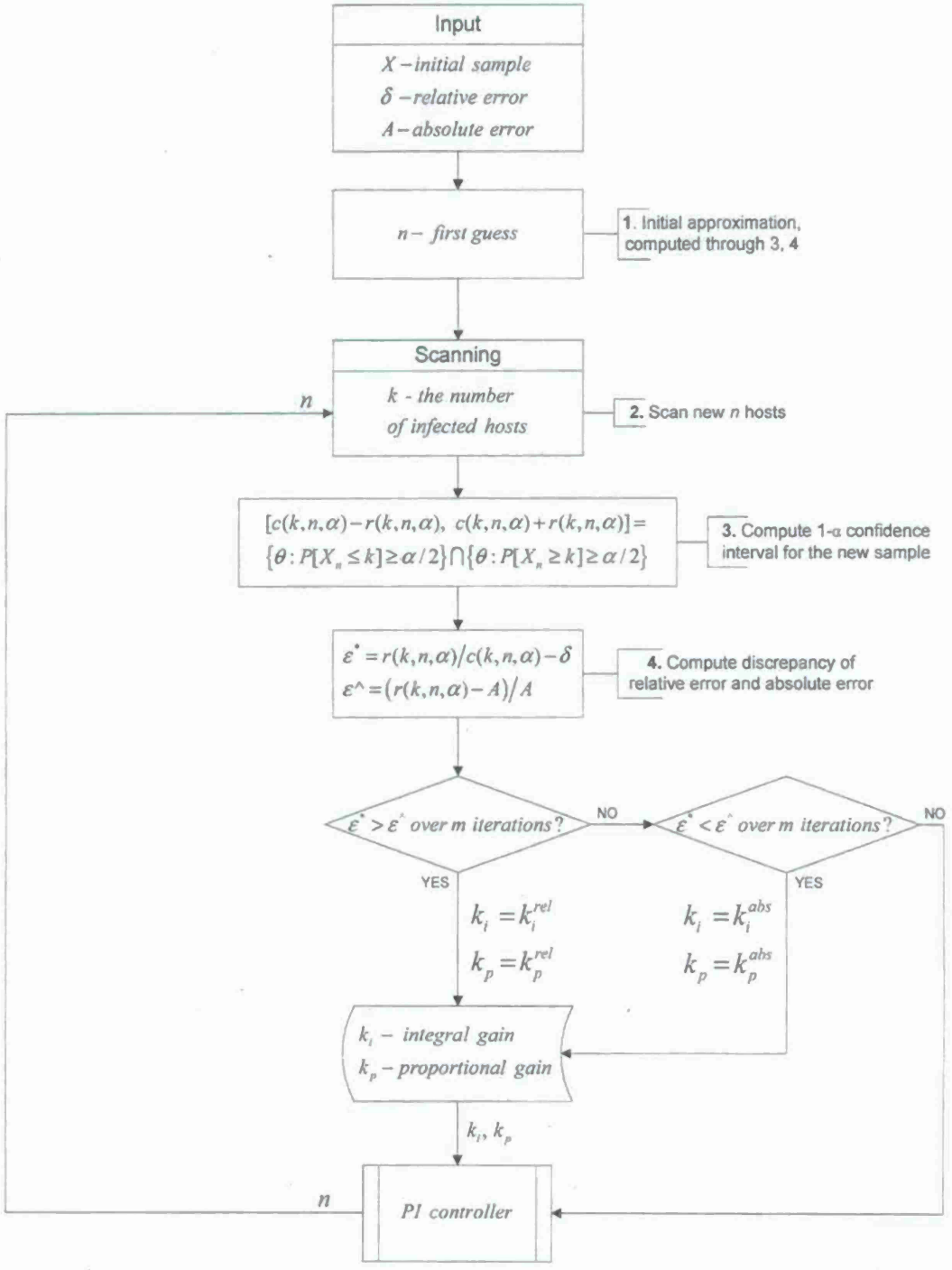


Figure 13: Estimation Error Control Procedure

6.4. Numerical Experiment

The numerical experiment was conducted to validate: the proposed technique by simulation of the complex interaction of worm, anti-worm and network bandwidth; the computation of the "true" number of infected hosts throughout this interaction; and the estimation of this number via the proposed approach. We used the discrete-time model presented in Section 3 to generate worm propagation dynamics.

The proposed estimation procedure was implemented in MATLAB. First we simulated the propagation of the worm using the discrete model and recorded worm population dynamics. Then we run the estimation procedure were we utilized binominal random number generator with proportion parameter set to the current number of infected hosts recorded in the trace array. In every time tick, the estimation procedure computed the sample size and random generator was used to simulate results of scanning. Figure 14 shows the results of the proposed procedure executed in trace-based simulation.

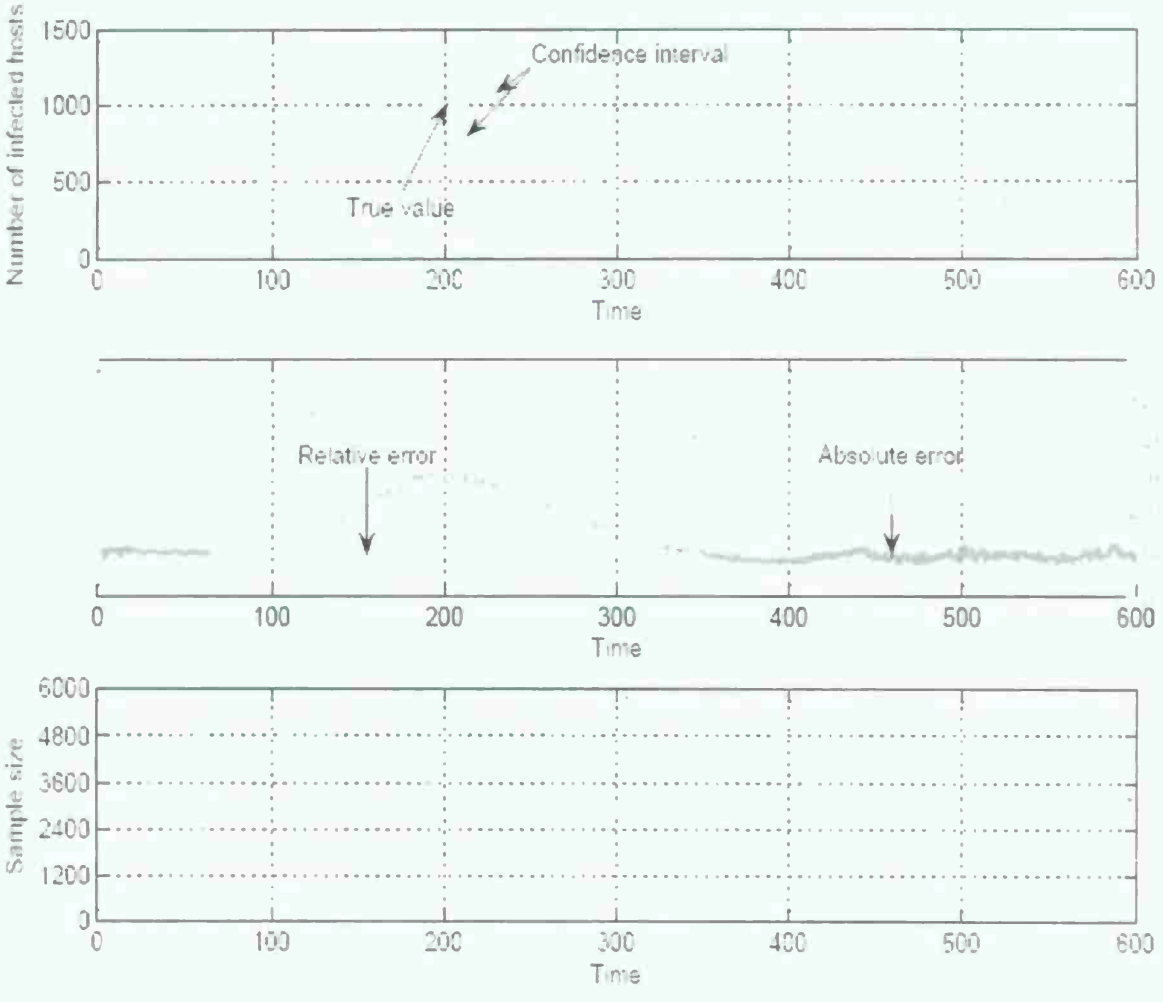


Figure 14: Dynamics of the Estimation Process

The relative error reference was set to 15% and the absolute error reference was set to 50 (hosts). The upper plot shows the confidence interval and the true value (in the middle) of the number of infected computers obtained from the model above. The middle plot illustrates relative error (left axis) and absolute error (right axis). The axis of the middle plot is scaled so that the references (0.15 to the left and 50 on the right axis) will geometrically coincide. The lower plot shows the sample size during each scanning session.

One can see that the true value of the infected hosts always remained within the confidence interval, indicating a consistent coverage probability for the exact interval. In the middle plot, the estimation error graphs have been marked out in bold in those sections where the particular error was controlled in the corresponding time period. We noted that for every instance (scanning session) whether relative error nearly converged to the reference (0.15) or absolute error converged to the reference (50). In fact, the output signal was switched from absolute to relative error at time tick 65 and back to absolute error at time tick 352. This result shows the proposed procedure's reliability and stability.

7.0 EXPERIMENTAL COMPUTER NERWORK FACILITY AT BINGHAMTON UNIVERSITY

The demand for higher degree of security is rapidly increasing within all levels of society, from individual computer users, to corporations and governments. Recently, cyber attacks on individual high profile targets within corporations, government and military infrastructures for the purpose of monetary profit, espionage and various destructive activities including global terrorism are on the rise. New attack schemes are being developed constantly including fast propagating internet worms carrying destructive payloads [1]. Scheduled backups and updated antivirus databases are no longer sufficient [71] for an adequate stand off against modern information attacks assuring uninterrupted network operation and data integrity. This situation nessisitates rapid analysis of information security threats in order to be able to achieve a significant level of success in fighting current attacks, as well as introduction of proactive measures to prevent future attacks [72]. Research in information assurance has significantly evolved over the past decade.

The most widely used approach to evaluate new network algorithms and their implementations for the Internet and Intranet environments are software network simulators [73] and small scale hardware node network testbeds. Evaluations of information security algorithms that do not require network connection or only require local connections within a small scale network are often performed in virtual environments such as Virtual PC and VMWare. Existing network simulators are rarely applicable to information security research and development, since many attack scenarios require the presence of actual vulnerable systems within the network for successful attack deployment and the following realistic network behavior analysis. Attack scenarios that are suitable for software network simulators often must be implemented according to specific requirements and topologies of the simulator that may differ from actual Internet environment, leading to less realistic experiments and questionable results. In addition, software simulators are not capable of emulating large scale network attack scenarios in real time at a rate one would expect from a physical network. On the other hand, physical network testbeds for information security experiments, while being closer to a real network topology and performance, lack the scalability of software network simulators and do not allow for rapid manipulation of both physical and software network resources, making it difficult to switch from one attack scenario to another.

The dependability of any experimental study in the area of information security research and development, hinges upon three main principles:

- Realistic network topology with real-time performance
- Rapid network topology modification and deployment
- Fast recovery after attack experiment

These principles require the evaluation of network attacks using actual nodes interconnected to form an Internet or Intranet network topology with flexibility of virtual environments for rapid modifications, deployment and recovery. This paper presents a design of a virtual computer network testbed utilizing both physical hardware and virtual software resources to enable rapid evaluation of new network security threats, new algorithms and methods for both retroactive and proactive attack countermeasures, as well as analysis and forecast of network performance under

various attack scenarios. The described test environment offers a realistic experience and performance of a physical hardware network, the flexibility of a software network emulator, and fast state recovery of a virtual machine in a lab environment.

7.1. Hardware Lab Design

Hardware infrastructure design is critical for any network security testbed. Depending on the overall design of the lab's topology, its hardware component can be chosen to meet specific needs of the lab. Designing a versatile network testbed based on virtual environments helps lowering the costs of hardware equipment, while enhancing the overall performance and resource utilization within the facility. In addition, the lab is designed to allow remote online access to the testing facility. Therefore, certain investments should be made in acquiring hardware that would provide an adequate security level for the lab infrastructure. The hardware acquisition was funded by a $185,000 Air Force DURIP grant. This section outlines the hardware component of the testbed.

Scalable computer network facilities

7.1.1. Scalable Computer Network Facilities

The controlled physical environment provides a safe, realistic and flexible foundation on which virtual networks can be constructed. The physical environment should be built and preconfigured to meet certain specification. The authors have defined the following minimal set of general specifications for an information security research and development testbed hardware infrastructure:

- High performance enterprise class servers with sufficient resources to handle multiple virtual appliances
- Dual network interfaces to support dedicated physical networks for malicious traffic and network management traffic
- Centralized storage for virtual components
- Dedicated terminals for network control and observation
- Dedicated security appliances to support network traffic filtering and secure remote lab management
- Centralized storage for testbed software backup images.
- Restricted physical access to testbed hardware component

Virtual machines (VM) allow for very efficient hardware resources utilization, while creating a scalable heterogeneous network environment. A reasonably powerful hardware server is capable of running 10 – 20 modern operating systems depending on OS type and resource hardware resource requirements. Due to the nature of network security research, operating system requirements per virtual machine can be pushed to the lower limit to accommodate a wider range of OS's running on the same physical server.

Virtual machines comprising the software backbone of the information security testbed require configuration and management. The testbed is designed to be configured both locally and remotely through management consoles and observation stations. At the same time the sensitive nature of the experimental studies to be conducted requires physical isolation of malicious traffic

from management traffic to prevent physical network and the external infrastructure from being compromised by malicious activity, and also to assure uncontaminated malicious traffic and network performance data. Figure 15 demonstrates the concept of dual interface for test lab management and experiments.

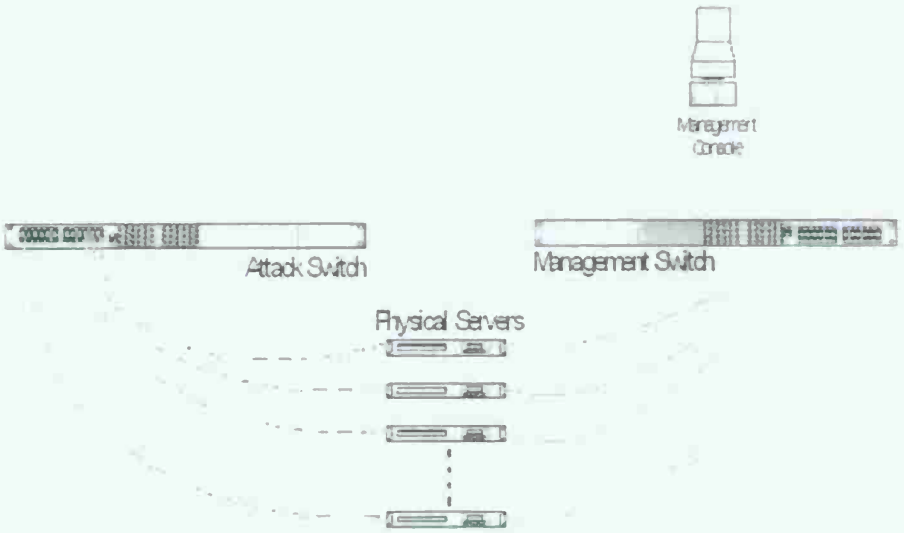


Figure 15 Dual Network Interface for concurrent execution of testbed experiments and lab management

An efficient virtual testbed requires a high degree of scalability. This implies the ability to choose various virtual components according to a predefined scenario of an experiment, and deploy these components onto the hardware framework. Large-scale network security experiments involve a large number of virtual components, distributed across multiple hardware nodes. In this situation, utilizing local server storage for storing all variations of virtual components is not sufficient, nor is it adequate for a large scale network testbed. A centralized storage facility, such as Network Attached Storage unit (NAS), for various virtual components enables network wide access to all available virtual components from any physical servers. Such a centralized storage should be efficiently protected from the experimental network to avoid malicious contamination. Therefore, it should be physically separated from the attack switch by having the storage facility connected only to the management network.

The network testbed is built to sustain network attack experiments of various nature, including real malware and exploits bearing harmful payloads. Although, most outbreaks of attacks will be contained within virtual environments that can be restored to their original state immediately at run time, operating systems hosting these virtual environments may also be corrupted due to improper virtual component handling and other issues. In order to assure rapid deployment and utilization of all available hardware units, it is highly desirable to have a centralized storage facility for backup images of host operating systems and their configurations.

7.1.2. Secure Network Administration

The security network testbed is designed to be accessible remotely to enable on demand reconfiguration, deployment and experimentation by multiple users. Therefore, the network security measures must prevent any accidental connections or packets reaching the WAN or the Internet. As it has been mentioned, the initial precaution is to physically separate malicious traffic from management traffic by introducing dedicated network switches. The next step is to prevent any accidental connections or packets from reaching the WAN or the Internet. Network security measures that must be in place include: default routes, router ACLs, network configuration, network monitoring, and the lab Firewall.

Default routes are facilitated by the router; the appliance has a default route to its internal interface, thereby stopping any traffic not explicitly bound for networks, not connected to the router interface. Consequently, any packets intended for servers not connected to the network testbed would not reach their intended destination. The router ACLs will impose an access rule on all traffic attempting to exit its primary interface. This access rule will effectively block all traffic that is not explicitly destined for the user accounts server. This ACL will prevent any Internet traffic from going beyond the router.

The user accounts server network configuration includes two networking interfaces for communicating with the testbed, and one interface for communicating with the firewall appliance. No other communication will exist within the user accounts server.

The main Firewall of the network testbed is represented by Firewall appliance with unlimited number of clients to facilitate incoming user administration. This Firewall provides network address translation to the Internet and the WAN only for the user accounts server; it must deny all unauthorized inbound initiated connections. Any traffic inconsistent with the acceptable traffic will be logged and disallowed. The complete physical network topology is shown in Figure 16.

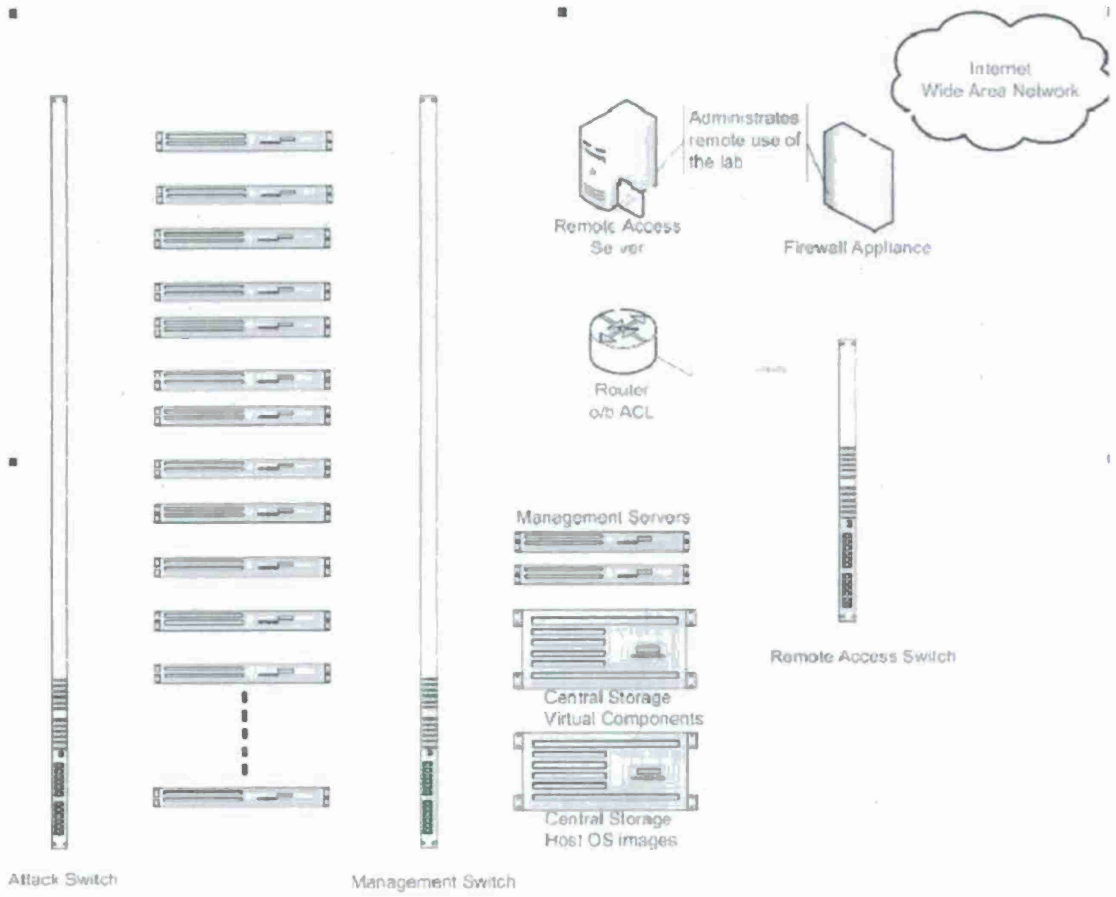


Figure 16 Hardware Testbed Topology

7.2. Software Lab Design

An efficient network security testbed includes a large variety of systems that are subjected to research and analysis. Such systems include both hardware and software components. In the past, information security testbeds contained a comparable number of both components, which led to inefficient testbed management and high costs of ownership. Current technology enables higher hardware utilization through software virtualization of various hardware components. This in turn allows researchers to have complete centralized control of the testbed, as well as the ability to modify the topology of the network without intruding testbeds' hardware. This section describes the design of the network, which supports virtual network infrastructure management for information security research and analysis.

7.2.1. Design of the Testbed Management Model

Modern network security research and development demands testbed facilities, designed to handle a larger number of security experiments in a shorter time. Most current network security

threats propagate and attack large-scale networks in a matter minutes causing significant damage. New security threats emerge quickly, requiring researchers and security analysts to respond faster in order to keep up with these attacks. New approaches for network security analysis, reactive and proactive network defense mechanisms should be evaluated under various network topologies to prove their effectiveness. Conventional hardware based network security testbeds always have certain tradeoffs between network scalability and deployment cycle.

On the contrary, virtual networks allow for practically unlimited variations of network topologies, while delivering on demand network topology deployment cycle. To demonstrate a simplified example of virtual testbed capabilities, consider the design of a typical hardware based network security testbed shown in Figure 17.

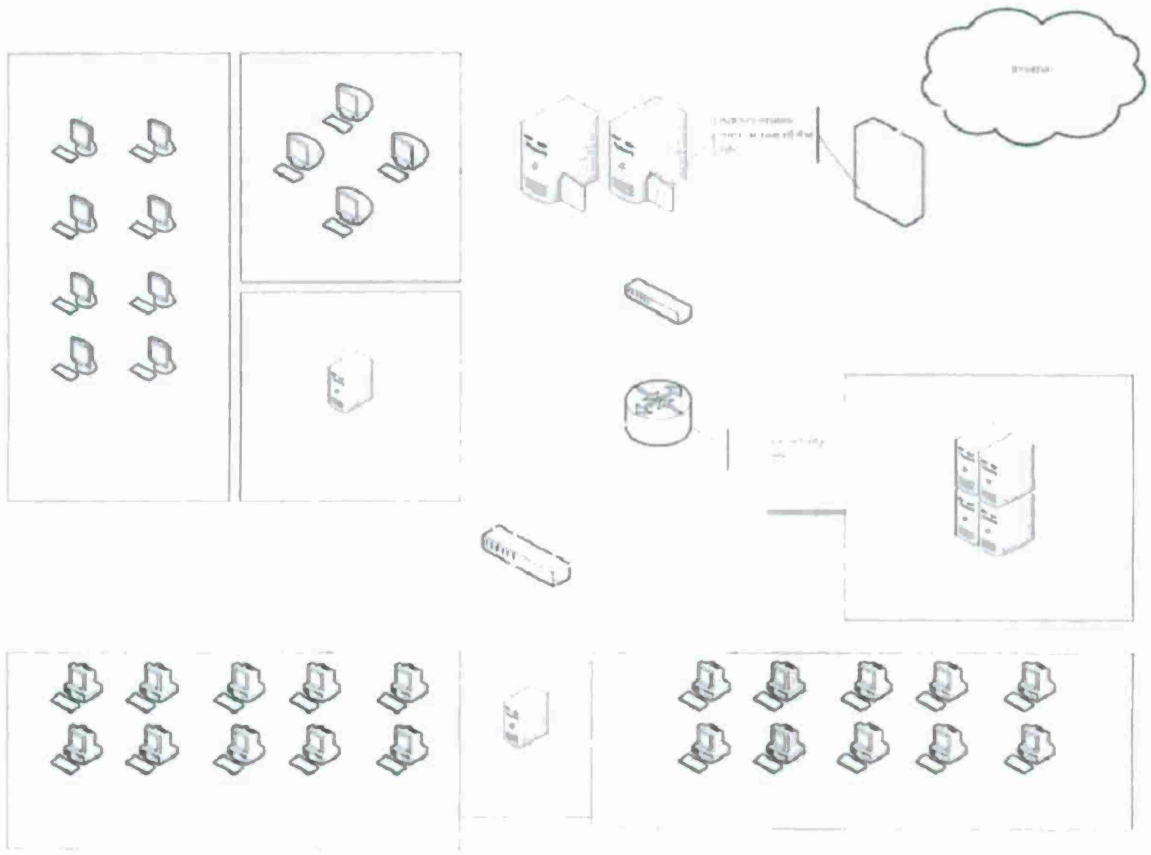


Figure 17: Typical Network Security Hardware-based Testbed

The way it is designed, this testbed is capable of handling large spectrum of network security experiments. Assuming that each physical node in this topology is a host for multiple virtual devices, such as operating systems, servers, switches, routers, etc. (Figure 18), this design can be converted into a new topology, as shown in Figure 19, significantly faster than in a conventional hardware based testbed without requiring extra equipment.

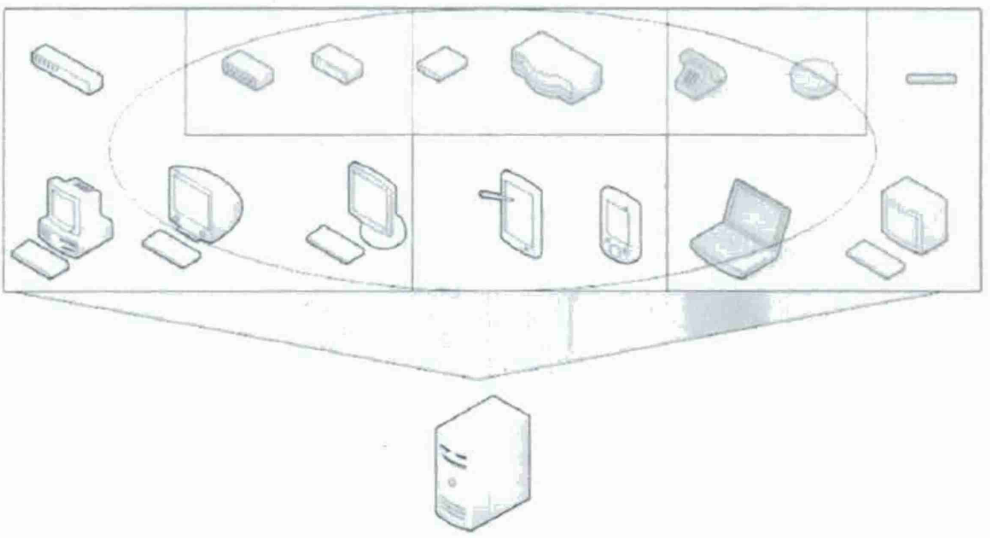


Figure 18: Virtual Network within a Single Hardware Node

The entire testing facility of the network security virtual testbed is constructed of virtual devices. All virtual machines are deployed on a set of physical servers, powerful enough to handle large CPU, memory and network bandwidth loads. Furthermore, all virtual machines are stored in a single storage container; they can be configured and deployed to create various network topologies. The capacity of the physical network backbone can handle hundreds of virtual devices running at the same time. A significantly larger number of virtual appliances are required to support various network topologies for information security research; one reason is because there may be different security vulnerabilities in different versions of same software package installed in a virtual machine. A prebuilt set of such virtual machines is required to have dynamic reconfigurable test environments within the testbed.

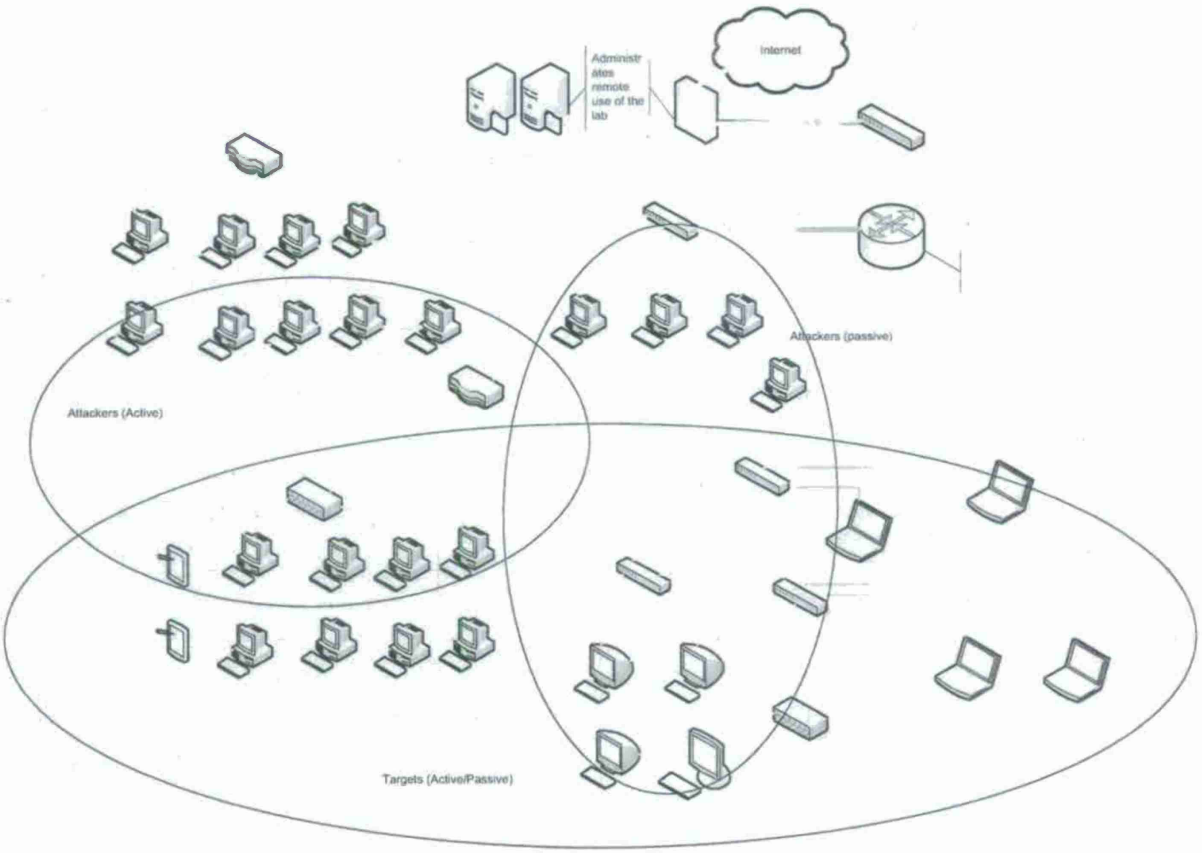


Figure 19: New Network Topology on Two Physical Nodes

The set of virtual appliances should be managed from a centralized, possibly remote location. Our management model requires several levels of software comprising the structure of the network security virtual testbed environment. The software management stack is shown in Figure 20. The host operating system consumes hardware resources when running virtual appliances. It is crucial to have the lightest possible host operating system in order to preserve valuable hardware resources for virtual components. Every virtual component is configured to have a limited set of resources, supplied by the hardware node. Due to potentially harmful malicious traffic flow within the network testbed, virtual appliances should be restricted from utilizing any means of network communications other than a dedicated malicious attack, in accordance with the testbed hardware configuration (Figure 16). The host operating system, having access to all virtual appliances it hosts, should collect information regarding their current state, resources consumption and network traffic logs, and report them to the Virtual Network Management Server. The server is responsible for management and configuration of virtual machines within multiple physical nodes of the testbed.

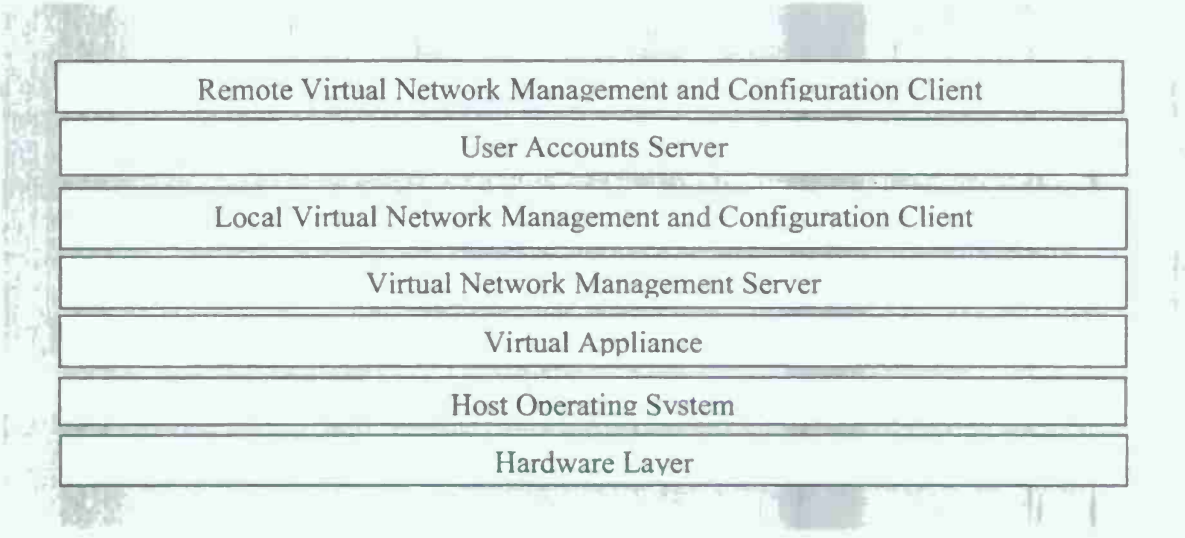


Figure 20: Network Security Testbed Management Software Stack

The next step in testbed management is to allow clients to control and configure the network properly. Due to security concerns and sensitive nature of the testbed, at least two levels of client management is established. We differentiate local and remote client access to the testbed, where the local client has higher privileges compared to the remote client. The local management client is only accessible within the physical facility. It is allowed to do the following operations on the security testbed:

- Access managed switches for both malicious and management networks
- Perform configuration of physical nodes
- Manage the pool of host operating system images
- Backup and restore host operating systems
- Manage the pool of virtual appliances

The remote management client has access to control and configuration of the pool or a subset of virtual appliances to perform network security experiments. The following is the list of operations performed by the remote client:

- Limited management of the pool of virtual appliances
- Configuration of the virtual network topology
- Deployment of the virtual network topology to reserved physical node
- Setup and execution of network security experiments on the created virtual network topology
- Observation, collection and analysis of information resulted from the experiment

It is possible to configure a remote management client account to allow performing actions restricted only to the local facility clients. In this case additional account management layer should be introduced to facilitate elevated access rights for the remote lab manager.

7.2.2. Design of Network Security Experiments

Network security experiments require large scale network testbeds. The virtual testbed is organized such that rapid deployment of various large-scale experiment scenarios is possible. The testbed provides important information for the computer network security research including analysis of malicious software, vulnerability analysis of network components, efficiency and dependability of security mechanisms, and network administration under attacks. The availability of a dedicated virtual experimental testbed allows researchers to deploy real information attacks and defense mechanisms with close monitoring of the status of the network. Virtual network experiment design is coherent with known techniques for simulated networks experiments and hardware based virtual networks [74]. The major advantage over known experiment design approaches is the ability to rapidly configure complex network topologies without direct access to hardware facility. The other advantage is the ability to choose various network components including operating systems, routers, servers, switches, etc., from a large centralized pool of preconfigured virtual appliances.

In order to facilitate an internet worm propagation experiment, the researchers have to configure the virtual network according to their experiment specifications [75]. Generally, specifications for worm propagation experiments include:

- The ability to reproduce the worm propagation in high fidelity
- Assessment of the impact of worm propagation on large scale networks
- Introduce worm detection defense mechanism
- Assessment of the impact of the detection mechanism on the network

Typically, network worm propagation rate greatly depends on availability of vulnerable systems compatible with the worm's exploit vector. Worms may carry destructive payloads, deployable on a vulnerable system after successful penetration. Often, the propagation engine of a worm heavily depends on success of its payload, further reducing propagation rate due to incompatible systems. A realistic worm propagation experiment on a virtual network testbed should take advantage of generating a large variety of susceptible operating systems and their configurations. Rapid adjustments to virtual network topology allows for fast elimination of invulnerable hosts to increase the rate of replication. This also advances realistic network damage assessment due to increased worm propagation rate. In the virtual network testbed environment, the researcher can visually select virtual appliances to be a part of the experiment, and connect these appliances to form a unique topology. A simplified virtual network topology including various operating systems connected through virtual switches operating on three hardware nodes is shown in Figure 21.

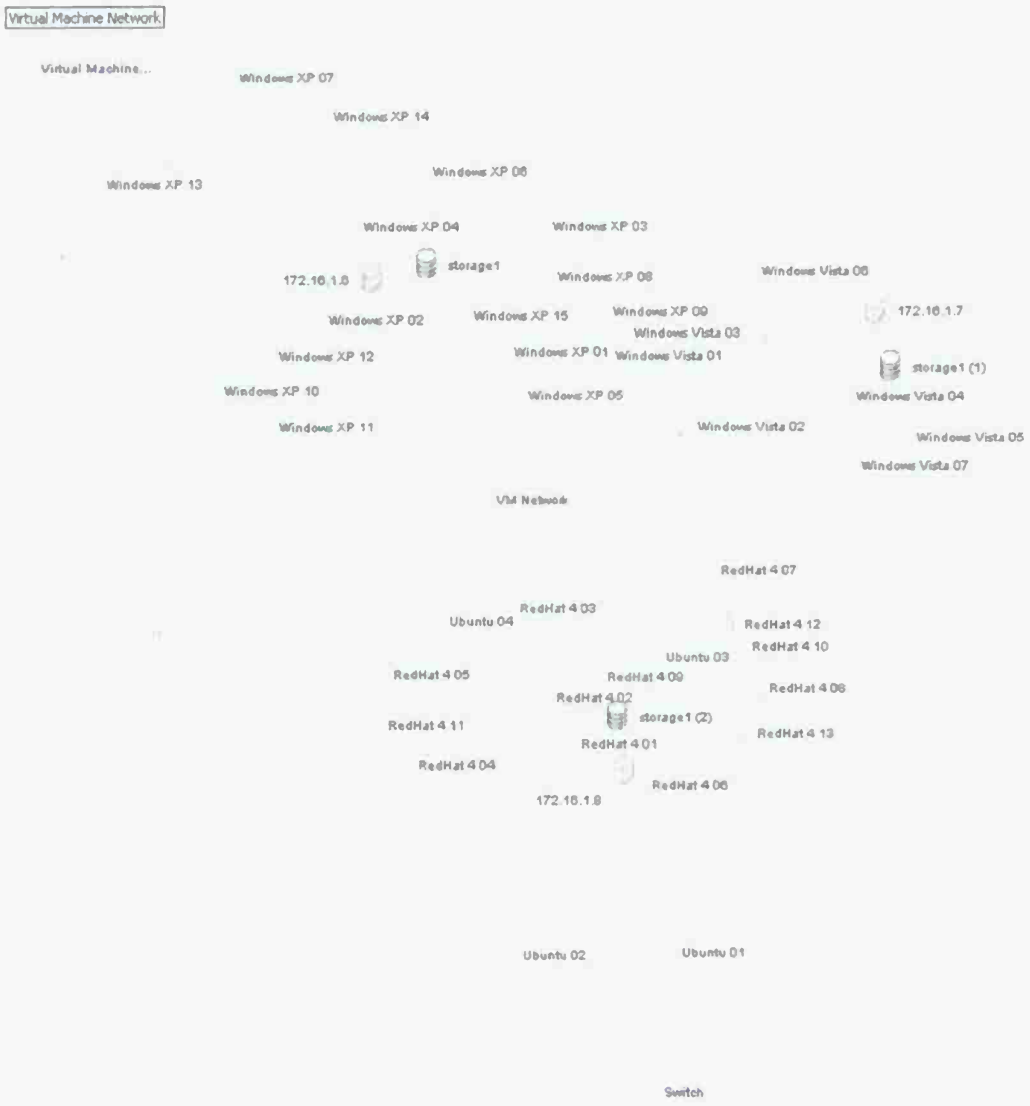


Figure 21: Virtual Network Topology for Worm Propagation Experiment Generated on 3 Hardware Nodes

Network damage assessment is performed by capturing required information from each virtual machine involved in the experiment. The information is collected from the management network (Figure 16), which controls all physical host, and therefore virtual appliances running on them. Malicious traffic may also be safely captured from special 1000 Mbps network ports, available on managed malicious traffic switches.

Further development of the experiment depends on the need to introduce a worm detection and defense mechanism into the network. If it is required to use the exact same topology, which has been successfully tested against worm propagation, for a proactive worm defense mechanism, the entire state of all virtual nodes within the network can be restored to its original uninfected condition. Otherwise, the worm defense agent can be introduced into an infected network.

8.0 CONCLUSION

The development of a fully automatic computer network security system capable of timely detection and mitigation of information attacks perpetrated by self-replicating malicious software is justified. Modern immunology presents a comprehensive example of such a system and demonstrates its major principles of operation. An extensive literature review provides background for the formulation of these principles as applied to a computer network. A configuration of a fully automatic computer network defense mechanism implementing active immune-type response is proposed.

Mathematical modeling of active immune-type response is a necessary basis for the deployment of the biologically-inspired defense mechanisms for computer networks. Such a model is formulated as a set of nonlinear equations describing complex interactions between three key factors representing the status of a computer network: the remaining amount of the network bandwidth, the number of infected hosts and the number of deployed self-replicating defense agents. A continuous-time and discrete-time models have been developed and utilized for the simulation of active immune-type processes in a computer network.

The concept of anti-worm, a self-replicating highly specialized software module is formulated. Various issues addressing feasibility, control of the propagation rate, discriminatory ability of the anti-worm are discussed; various solutions are proposed and supported by known literature.

The authors applied modern automatic control theory for the development of a negative feedback control mechanism responsible for the propagation rate of the deployed anti-worm. This mechanism assures that the resultant active immune-type process in the computer network is globally asymptotically stable, that could be interpreted as the ability of the network to recover form the information attacks perpetrated by computer worm.

A technology for selective sampling (scanning) of the network hosts and statistical analysis of the scanning results for the network status assessment is developed and tested using by computer simulation. While the "actual", i.e. simulated status of the network is known beforehand, the developed approach showed its ability to monitor the network status with the required degree of statistical confidence. A trade-off between the impact on the network bandwidth and the confidence of the status estimation is demonstrated.

Based on the previously developed concept of dynamic code analyzer (DCA), the authors developed and evaluated Propagation Engine Detector (PED) system capable of detecting attacks perpetrated by network worms. This system detects the worm shell code activity performed by a process during the attack session. Moreover, PED recognizes the type of propagation engine employed by the worm exploiting the process. The developed PED system utilizes Colored Petri nets (CP-net) to trace in parallel interrelated chains of system calls issued by the monitored process to recognize high level API functions invoked by the process. The detected high level functions in combination are analyzed to determine how the process creates and manipulates operation system objects. Such information is finally processed by CP-net to detect if the process activity exhibits the functionality of the particular type of propagation engine.

While experimental work with "live" malware is potentially dangerous, a safe experimental facility is developed for the experimental implementation and validation of the presented results. This is perceived as the further development of this research and result in a new paradigm in computer network security by employing immune defenses honed to perfection by million-year evolution to assure safety and dependability of future computer networks.

9.0 REFERENCES

1. Victor A. Skormin, *Biological Approach to System Information Security (BASIS) A New Paradigm in Autonomic Information Assurance.* Final report to AFRL on contract #30602-01-0509, Binghamton NY, 2002,

2. Skormin, V.A., Delgado-Frias, J.G., McGee, D.L., Giordano, J.V., Popyack, L.J., Gorodetski, V.I. and Tarakanov, A.O., "BASIS: a biological approach to system information security, Information Assurance in Computer Networks" *LNCS 2052,* Springer-Verlag, Berlin, 2001, pp.127-142.

3. Kephart J. O., *A Biologically Inspired Immune System for Computers*, IBM Thomas J. Watson Research Center, High Integrity Computing Laboratory, 1994.

4. Kephart J. O., White S. R. "Directed–graph epidemiological models of computer viruses", *Proceedings of the IEEE Symposium on Security and Privacy*, 1991, pp. 343–361.

5. Kephart J. O., White S. R. "Measuring and Modeling Computer Virus Prevalence", *Proceedings of the IEEE Symposium on Security and Privacy*, 1993.

6. Kephart J. O., White S. R., Chess S.R. "Computers and Epidemiology", *IEEE Spectrum*, May 1993, pp.20–26.

7. Kephart J. O.: "How topology affects population dynamics." *C. Langton, ed., Artificial Life III. Studies in the Sciences of Complexity.* 1994, pp. 447 – 463.

8. Kephart J., Sorkin G., Chess D., White S., "Fighting Computer Viruses", *Scientific American,* November 1997.

9. Kephart J., Sorkin G., Chess D., Swimmer M., White S., "Blueprint for a Computer Immune System", *The Virus Bulletin International Conference in San Francisco*, California, October 1–3, 1997.

10. Zou C., Gong W., Towsley D., "Code Red Worm Propagation Modeling and Analysis", *In 9th ACM Conference on Computer and Communication Security*, Washington DC, 2002.

11. Zou C., Gong W., Towsley D., "Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense", *WORM'03*, October 27, 2003.

12. Daley D.J., Gani. J., **Epidemic Modelling: An Introduction.** Cambridge University Press, 1999.

13. Nicol D., Liljenstam M., *Comparing Passive and Active Worm Defenses* Coordinated Science Laboratory, University of Illinois, 2000.

14. Nicol D., Liljenstam M., *Models of Active Worm Defenses,* Coordinated Science Laboratory, University of Illinois 2004.

15. Kim J., Radhakrishnan S., Dhall S., *Measurement and analysis of worm propagation on Internet Network Topology*, School of Computer Science, University of Oklahoma, Norman, Oklahoma, USA. 2003.

16. Wang Y., Chakrabati D., Wang C., Faloutsos C., "Epidemic spreading in real networks: an Eigen value viewpoint", *Proceedings of 22nd International Symposium on Reliable Distributed Systems,* October, 2003.

17. Staniford S., Paxson V., Weaver N., "How to own the internet in your spare time", *In Proceedings of the 11th USENIX Security Symposium,* August, 2002.

18. Pastor–Satorras R. and Vespignani A., "Epidemic dynamics and endemic states in complex networks", *Physical Review E,* 63:066117, 2001.

19. Pastor–Satorras R., Vespignani A., "Epidemic spreading in scale–free networks", *Physical Review Letters*, 86(14): pp.3200–3203, 2 April 2001.

20. Pastor–Satorras R., Vespignani A., "Epidemic dynamics infinite size scale–free networks", *Physical Review E*, 65:035108, 2002.

21. Pastor–Satorras R., Vespignani A., "Epidemics and immunization in scale–free networks", In *S. Bornholdt and H. G. Schuster, editors, Hand–book of Graphs and Networks: From the Genome to the Internet.* Wiley–VCH, Berlin, May 2002.

22. Pastor–Satorras R., Vespignani A., "Immunization of complex networks", *Physical Review* E, 65:036104, 2002.

23. Boguna M., Pastor–Satorras R., "Epidemic spreading in correlated complex networks", *Physical Review* E, 66:047104, 2002.

24. Barabasi A.–L., Albert R., "Emergence of scaling in random networks", *Science,* 286:509–512, 15October 1999.

25. Wang C., Knight J. C., Elder M. C., "On Computer Viral Infection and the Effect of Immunization", *Proceedings of the 16th Annual Computer Security Applications Conference,* 2000, pp. 246–256.

26. Castañeda F., Sezer E., Xu J., "WORM vs. WORM: Preliminary Study of an Active Counter-Attack Mechanism" *Proceedings of WORM'04,* October 2004

27. Sidiroglou, S., Keromytis, A.D., "Countering network worms through automatic patch generation" *IEEE Security and Privacy*, 2005.

28. Liang, Z., Sekar, R., "Fast and automated generation of attack signatures: A basis for building self-protecting servers" *Proceedings of CCS*, 2005

29. Liang, Z., Sekar, R., "Automatic Generation of Buffer Overflow Attack Signatures: An Approach Based on Program Behavior Models" *Proceedings of ACSAC* 2005

30. White S.R., Swimmer M., Pring E.J., Arnold W.C., Chess D.M., Morar J.F., *Anatomy of a Commercial–Grade Immune System*, IBM Thomas J. Watson Research Center, 1997.

31. Liljenstam M., Nicol D., Berk V., Gray B., "Simulating realistic network worm traffic for worm warning system design and testing", *In proceedings of the First ACM Workshop on RapiMalcode.* 2003.

32. Moore D., Shannon C., Voelker G., Savage S., "Internet quarantine: Requirements for containing self–propagating code", *In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*. 2003.

33. Zou C., Gao L., Gong W., Towsley D., "Monitoring and early warning for internet worms", *In Proceedings of 10th ACM Conference on Computer and Communication Security.* 2003.

34. Stengel R., Ghigliazza R., Kulkarni N., "Optimal enhancement of immune response", *Bioinformatics, **Vol. 18**, N9,* 2002, pp. 1227–1235.

35. Stengel R.F., Ghigliazza R., "Stochastic optimal therapy for enhanced immune response", *Mathematical Biosciences,* N191, 2004, pp. 123–142.

36. Stengel R. F., Ghigliazza R., Kulkarni N., Laplace O., "Optimal control of innate immune response", *Opt. Control Appl. & Meth.,* N23, 2002, pp.91–104.

37. Swierniak A., Ledzewicz U., Schattler S., "Optimal Control for a Class of Compartmental Models in Cancer Chemotherapy", *Int. J. Appl. Math. Comput. Sci.,* N13, 2003, pp. 101–112.

38. Castiglione F., Piccoli B., *Optimal control in a model of dendritic cell transfection cancer immunotherapy,* Istituto Applicazioni del Calcolo (IAC), M. Picone, Consiglio Nazionale delle Ricerche (CNR), Italy, August 27, 2004.

39. Culshaw R.,· Ruan S., · Spiteri R., "Optimal HIV treatment by maximizing immune response*", J. Math. Biol.* 48, 545–562, 2004.

40. Joshi H. R., *Optimal Control of an HIV Immunology Model,* Department of Mathematics University of Tennessee Knoxville, 1999.

41. Berman, S. M., "Optimal Timing of Antiviral Therapy in HIV Infection", *J. Appl. Prob.*, 31A, 3–15, 1994.

42. Butler, S., Kirschner, D. and Lenhart, S., "Optimal Control of the Chemotherapy Affectingthe Infectivity of HIV", *Mathematical Biology and Medicine,* **Vol. 6**, World Scientific, 1995.

43. Fister, K. R., Lenhart, S. and Mc Nally, J. S., "Optimizing Chemotherapy in an HIV Model", *Electron. J. Diff. Eqns.*, N32, 1998, pp.1–12,.

44. Ledzewicz U., Schattler H., "Optimal Control for a Bilinear Model with Recruiting Agent in Cancer Chemotherapy", National Science Foundation under grants No. 0205093 and No. 0305965, 2003.

45. Ledzewicz U., Schattler H., "Optimal bang–bang controls for a 2–compartment model in cancer chemotherapy" Journal *of Optimization Theory and Applications* – JOTA, 114, 2002, pp. 609–637.

46. Ledzewicz U., Schattler H., "Analysis of a cell–cycle specific model for cancer chemotherapy", *J. of Biological Systems*, 10, 2002, pp. 183–206

47. Swierniak A., "Cell cycle as an object of control", *Journal of Biological Systems*, 3, 1995, pp. 41–54.

48. Swierniak A., Duda Z., "Bilinear models of cancer chemotherapy–singularity of optimal solutions", *Mathematical Population Dynamics*, N2, 1995, pp. 347–358.

49. Swierniak A., Polanski A., Kimmel M., "Optimal control problems arising in cell–cycle–specific cancer chemotherapy", *Cell prolif.*, N29, 1996, pp. 117–139.

50. Pontryagin L. S., Boltyanskii V. G., Gamkrelidze R. V., Mishchenko E. F., **The mathematical theory of optimal processes, Vol 4**, Gordon and Breach Science Publishers, 1986.

51. Knowles G., **An Introduction to Applied Optimal Control**, New York, Academic Press, 1981.

52. Bertsekas D.P., **Dynamic Programming and Optimal Control, Vol. II**, 2nd Edition, Athena Scientific, Belmont, MA, 2001.

53. Bertsekas D.P., **Dynamic Programming and Optimal Control, Vol. I**, 3rd Edition, Athena Scienti.c, Belmont, MA, 2005.

54. Kim J., Radhakrishnan S., Dhall S., *Optimal Control of Treatment Costs for Internet Worm,* School of Computer Science, University of Oklahoma, Norman, Oklahoma, USA.

55. Goldman S.M., Lightwood J., "Cost Optimization in the SIS Model of Infectious Disease with Treatment", *The B.E. Journals in Economic Analysis and Policy*, Vol.2. No.1 1995.

56. Volynkin, A., Skormin, V., Summerville, D., Moronski J., Evaluation of Run-Time Detection of Self-Replication in Binary Executable Malware. In*: Proceedings of the 7th IEEE Systems, Man and Cybernetics Information Assurance Workshop* June 2006

57. Hofmeyr S. A., Forrest S., Somayaji A., "Intrusion detection using sequences of system calls, " *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

58. Durante A., Pietro R. Di, Mancini L. V., "Formal Specification for Fast Automatic IDS Training" *Lecture Notes in Computer Science*, 2629:191-204, 2003.

59. Stolfo S., Lee W., Eskin E., "Modeling system calls for ID with Dynamic Window Sizes", *In Proceedings of the DISCEX II*. June 2001.

60. Liu A., Martin C., "A Comparison of System Call Feature Representations for Insider Threat Detection". *In Proceedings of the 6th IEEE Information Assurance Workshop*, 2005.

61. Tandon G., Chan P., "Learning Useful System Call Attributes for Anomaly Detection," *In Proceedings of the FLAIRS Conference*, 2005.

62. Xu M., Chen C., Ying J., "Anomaly detection based on system call classification," *Journal of Software*, 15(3): pp. 391-403, 2004.

63. Kruegel C., Mutz D., Valeur F., Vigna G., "On the Detection of Anomalous System Call Arguments," *ESORICS*, Oct. 2003.

64. Bernaschi M., Grabrielli E., Mancini L., "Operating System Enhancements to Prevent the Misuse of System Calls", *In Proceedings of the ACM Conference on Computer and Communications Security*, pp. 174 – 183, 2000.

65. Kang D., Fuller D., Honavar V., "Learning classifiers for misuse and anomaly detection using a bag of system calls representation". *In Proceedings of 6th IEEE Systems Man and Cybernetics Information Assurance Workshop (IAW)*, pp. 118-125, 2005.

66. Bowen T., Segal M., Sekar R., "On preventing intrusions by process behavior monitoring". *In Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, 1999.

67. www.metasploit.com

68. Jensen Kurt, **Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use**, volume 1, Springer-Verlag, Berlin, 1996

69. Volynkin A., Skormin V., "Large-scale Reconfigurable Virtual Testbed for Information Security Experiments," *in Proceedings of the 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*,

70. Lawrence D., Brown T., Cai Tony, DasGupta Anirban, "Interval Estimation for a Binomial Proportion," *Statistical Science*, **vol. 16**, number 2 (May, 2001), pages 101-1174.

71. McGraw G., "Managing Software Security Risks," *Computer* 35-4, 2002, pp. 99-101.

72. Gorodetski V., Skormin V., Delgado-Frias J., McGee D., Giordano J., Popyack L., Tarakanov A., "BASIS: A Biological Approach to System Information Security,"

Proceedings of the International Workshop Mathematical Methods, Models and Architectures for Computer Network Security, Lecture Notes in Computer Science, Springer Verlag, Saint-Peresburg, Russia, 2001, pp.127-142

73. McCanne S., Floyd S., "Ns-2 Network Simulator," http://www.isi.edu/nsnam/ns/, 1997

74. Benzel T., Braden B., Kim D., Neuman C., Joseph A., Sklower K., Ostrenga R., Schwab S., "Experience with DETER: A Testbed for Security Research," *2nd IEEE Conference on testbeds and Research Infrastructures for the Development of Networks and Communities*, Spain, 2006.

75. Miyachi Toshiyuki, Chinen Ken-ichi, Shinoda Yoichi, "Automatic Configuration and Execution of Internet Experiments on an Actual Node-based Testbed," *1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, IEEE, pp. 274-282, Trento, Italy, 2005.

10.0 LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

AAWP	Analytical Active Worm Propagation model
ACL	Access Control List
AFRL	Air Force Research Laboratory
AIR	Active Immune Response
BASIS	Biological Approach to System Information Security
CERT/CC	Coordination Center of Reaction to Computer incidents
CP-net	Colored Petri nets
DCA	Dynamic Code Analyzer
ED&E	Executable download and Execute
FSM	Finite State Machine
HIV	Human Immunodeficiency Virus
IDS	Intrusion Detection System
NAS	Network Attached Storage unit
PRNG	Pseudo Random Number Generator
PED	Propagation Engine Detector
QoS	Quality of Service