ON COUNTERMEASURES OF WORM ATTACKS OVER THE INTERNET

A Dissertation

by

WEI YU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Computer Engineering

ON COUNTERMEASURES OF WORM ATTACKS OVER THE INTERNET

A Dissertation

by

WEI YU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee:	Wei Zhao
	Riccardo Bettati
Committee Members:	Narasimha Reddy
	Jennifer Welch
Head of Department:	Valerie E. Taylor

May 2008

Major Subject: Computer Engineering

ABSTRACT

On Countermeasures of Worm Attacks over the Internet.

(May 2008)

Wei Yu, B.S., Nanjing University of Technology;

M.S., Tongji University

Co-Chairs of Advisory Committee: Dr. Wei Zhao
Dr. Riccardo Bettati

Worm attacks have always been considered dangerous threats to the Internet since they can infect a large number of computers and consequently cause large-scale service disruptions and damage. Thus, research on modeling worm attacks, and defenses against them, have become vital to the field of computer and network security. This dissertation intends to systematically study two classes of countermeasures against worm attacks, known as traffic-based countermeasure and non-traffic based countermeasure. Traffic-based countermeasures are those whose means are limited to monitoring, collecting, and analyzing the traffic generated by worm attacks. Non-traffic based countermeasures do not have such limitations.

For the traffic-based countermeasures, we first consider the worm attack that adopts feedback loop-control mechanisms which make its overall propagation traffic behavior similar to background non-worm traffic and circumvent the detection. We also develop a novel spectrum-based scheme to achieve highly effective detection performance against such attacks. We then consider worm attacks that perform probing traffic in a stealthy manner to obtain the location

infrastructure of a defense system and introduce an information-theoretic based framework to obtain the limitations of such attacks and develop corresponding countermeasures.

For the non-traffic based countermeasures, we first consider new unseen worm attacks and develop the countermeasure based on mining the dynamic signature of worm programs' run-time execution. We then consider a generic worm attack that dynamically changes its propagation patterns and develops integrated countermeasures based on the attacker's contradicted objectives. Lastly, we consider the real-world system setting with multiple incoming worm attacks that collaborate by sharing the history of their interactions with the defender and develop a generic countermeasure based on establishing the defender's reputation of toughness in its repeated interactions with multiple incoming attackers to optimize the long-term defense performance.

This dissertation research has broad impacts on Internet worm research since this work is fundamental, practical and extensible. Our developed framework can be used by researchers to understand key features of other forms of new worm attacks and develop countermeasures against them.

DEDICATION

To MY FAMILY.

ACKNOWLEDGMENTS

To reach this stage in my Ph.D. study and this point in my life, I am indebted to many great people for their wisdom, support, and love.

I would like to express my deep appreciation to my advisor, Dr. Wei Zhao. I am greatly indebted to him for his constant inspiration, encouragement, and guidance throughout my Ph.D. studies, which were essential to the completion of this dissertation. His broad vision has always been the inspiration in my existing and future professional growth. I would like to thank my co-advisor, Dr. Riccardo Bettati, for his sound technical advice and encouragement of my research. I would like to thank Dr. A. L. Narasimha Reddy for his guidance as a member of my dissertation committee. His insightful advice of my research work has inspired me to achieve important results. Many thanks go to Dr. Jennifer Welch for providing great technical advice to my research and serving on my advisory committee. I would like to thank Dr. Donald Friesen for his kind advice relating to many academic issues.

Many thanks also go to former and current graduate students in the Real-Time Systems group. It was a great fortune for me to work with Dr. Dong Xuan and his Ph.D students who have provided me with useful assistance during my research. I have benefited greatly from working with Dr. Xinwen Fu and Dr. Nan Zhang over the past few years. I have also had many helpful discussions with Dr. Yong Guan, Dr. Shengquan Wang, Dr. Shu Jiang, Dr. Chengzhi Li, and Dr. Ye Zhu. I would like to express my appreciation to Ms. Elena Catelena, Ms. Larisa Archer, and Ms. Valerie Ann Sorenson for their friendship and help during my Ph.D. studies. In particular, I would like to thank Ms. Archer for her selfless help in polishing many of my writings. I would like to express my appreciation to my Cisco colleagues, Mr. Man Loh, Dr.

Ping Ni, Mr. Tony Zhu, Mr. Ed Chen, Mr. Lee Ji, Mr. Dagang Wang, Mr. Thang Nguyen, Mr. Clint Entrop, Mr. Chris Pearce and many others for their friendship and help.

I am indebted to my parents, parents-in-law, my sister, and her husband, and their lovely son for their unconditional love and support. I would like to take this opportunity to express my greatest gratitude to my wife, Jian Wang, for her selfless love and support these years. It is this family that has made me strong and courageous to be the person I am today.

TABLE OF CONTENTS

Page

Page

Page

LIST OF FIGURES

Page

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

1. Worm Attacks Are Major Threats to the Internet

Worm attacks have recently posed major threats to the Internet. For example, in July 2001, a worm called "Code-Red" infected more than 350,000 Microsoft servers running Internet information service (IIS). In less than 14 hours, this worm caused more than 1.2 billion dollars in economic damages [1]. In January 2003, another worm called "Slammer" infected nearly 75,000 Microsoft SQL servers in less than 10 minutes and consequently caused large scale disruptions in production systems worldwide [2]. In March 2004, worms called "Witty" and "Sasser" infected many computers in a short time, rendering them unusable [3].

Furthermore, a recent trend of worm attacks has emerged in the way used to launch subsequent attacks. For example, "Code-Red" worms launched the distributed denial-of-service (DDoS) attack against the White House's website (www.whitehouse.gov) at the final stage of their propagation [1]. In February 2004, the "MyDoom" worm propagated rapidly to many computers that launched the DDoS attack against numerous websites, such as www.sco.com and www.microsoft.com, thereby preventing legitimate users from accessing them [4]. In addition to DDoS attacks, recent studies have shown that a large number of infected computers have been used to form the botnet as a black-market incentive for trading and/or renting infected computers to launch other attacks [5, 6, 7, 8, 9]: (i) access confidential information that can be abused through large scale traffic sniffing, key logging, identity theft etc., (ii) distribute large scale unsolicited advertisement emails (as spam) or software (as adware), (iii) spread new malware by installing Trojan Horses or other backdoor software, and (iv) destroy data that has high monetary

---

This dissertation follows the style and format of *IEEE/ACM Transactions on Networking*.

value.

2. Overview of Dissertation Research

Due to the massive damage potentially caused by worm proliferation, research on modeling worm attacks, and defenses against them, have become vital to the field of computer and network security. This dissertation intends to systematically study two classes of countermeasures against worm attacks, known as traffic-based countermeasure and non-traffic based countermeasure. Traffic-based countermeasures are those that detect worm attacks by purely monitoring, collecting, and analyzing the traffic generated by worm attacks. Non-traffic based countermeasures are those that detect worm attacks without being limited to monitoring, collecting, and analyzing the traffic generated by worm attacks. After the attack is detected, subsequent schemes can be applied to mitigate the attack's effectiveness. For example, patches can be released to fix the vulnerability, worm attack traffic can be throttled and filtered, and infected computers can be quarantined and recovered [10, 11, 12, 13, 14, 15].

2.a. Traffic-Based Countermeasures

The first component of this dissertation research is to develop traffic-based countermeasures. In order to develop these types of countermeasures, we consider both simple and sophisticated attack models and consequently develop countermeasures based on two types of traffic generated by worm attacks. Specifically, for the simple model, a worm attack will generate propagation traffic (i.e., messages that intend to identify vulnerable computers) directly. For the sophisticated model, a worm attack will first attempt to generate probing messages in order to identify the location infrastracture of the defense system, thereby circumventing the detection. Based on propagation traffic and probing traffic, our traffic-based countermeasures consist of the following two components.

1) Countermeasure Based on Propagation Traffic: Considering worm attacks which adopt the feedback loop-control mechanisms to manipulate the propagation traffic in order to make it similar to the background traffic and circumvent the detection, we develop a novel spectrum-based scheme to defend against such attacks. Our design is based on the insight observation: while the worm propagation traffic and background traffic are barely distinguishable in the time domain, their distinction is clear in the frequency domain, due to the recurring manipulative nature of such worms. Our countermeasure scheme uses the Power Spectral Density (PSD) distribution of the propagation traffic rate and its corresponding Spectral Flatness Measure (SFM) to distinguish the worm propagation traffic from non-worm (background) traffic. Our evaluation data clearly demonstrate that our proposed scheme can effectively detect such worm attacks.

2) Countermeasure Based on Probing Traffic: Considering worm attacks which carry out probing traffic in a stealthy manner, e.g., launching low-rate of probing traffic encoded by Pseudo-Noise (PN) codes, we develop countermeasures against such attacks. Our analytical, simulation, and empirical data first demonstrate the feasibility of such low-rate probing attack in practice. To counteract such attacks, we then introduce an information-theoretical framework and map strategies for attacks to coding strategies for communication channels. We propose a countermeasure that monitors the traffic-rate change of an individual monitor in a time-series manner. We show that the power constraints enforced by the countermeasure can significantly reduce the channel capacity of a system to a fairly low level that practically eliminates localization attacks on ITM systems. Our data validates our findings and shows the effectiveness of our developed countermeasures in terms of meaningless prolonged time for the attackers to launch such attacks.

2.b. Non-Traffic Based Countermeasures

The second component of this dissertation research is to develop non-traffic based countermeasures, as supplementary approaches against worm attacks. In order to develop these types of countermeasures, it is critical to identify what types of non-traffic features must be related to the worm attack and understand their characteristics. Motivated by the fact that most existing research on this topic are either based on features of known worms or ones that can be easily manipulated, our work intends to develop countermeasures based on more robust features which are difficult to manipulate by worm attacks. To this end, based on worm uncontrollable features such as dynamic signature of worm program execution, attackers' contradicted objectives and the defender's reputation, our non-traffic based countermeasures consist of three parts, as follows:

1) Countermeasure Based on Dynamic Signature: Considering the new unseen worm attack, we propose a novel detection approach based on mining dynamic signatures of worm program run-time executions. Our approach allows for the capture of dynamic behavior of executables and provides accurate and efficient detection against both seen and new unseen worms. We execute a large number of real-world worms and benign executables and trace their system calls. Via mining signatures from a large amount of features extracted from the system call traces, we apply two classifier learning algorithms, known as *Naive Bayes* and *Support Vector Machine* (SVM). The learned classifiers are further used to carry out rapid worm detection with low overhead on the end-host. Our experimental results clearly demonstrate the effectiveness of our approach to detect worm attacks in terms of very high detection rate and low false positive rate.

2) Countermeasure Based on Contradicted Objectives: Taking into consideration that a worm attack becomes smarter and manipulates features used by countermeasures, we consider the fact that no matter how a worm attack changes strategies, one thing it cannot change is its objectives.

Based on this, we develop one novel non-traffic based countermeasure by testing an important non-traffic feature − contradicted objectives to defend against worm attacks. In particular, we develop the countermeasures against a general form of worms, referred to as self-adaptive worms that adapt their propagation patterns in order to reduce the probability of detection, and to eventually infect more computers. To develop proper countermeasures, we introduce a game-theoretic formulation to model the interaction between the worm propagator and the defender. We show that an effective integration of multiple countermeasure schemes (e.g., worm detection and forensics analysis) is critical for defending against self-adaptive worms, which can force the worm attacker to choose the contradicted objectives. We propose different integration of countermeasure schemes for different kinds of self-adaptive worms, and evaluate their performance via real-world traffic data.

3) Countermeasure Based on Defender's Reputation: Considering the real-world system settings with multiple incoming worm attackers that collaborate by sharing the history of their interactions with the defender, we propose a novel countermeasure based on establishing the defender's reputation of toughness in its repeated interactions with multiple incoming attackers. Our studies show that while such iterative attacks may enable an attacker to learn from previous interactions, the defender can also take advantage of the iteration by sacrificing short-term performance in the initial few rounds to establish a "tough" reputation, in return for much higher payoff in the long-run by using the established reputation to force subsequent attackers to drop their attacks. Our extensive theoretical analysis and numerical results based on the study of worm detection shows that our reputation-aware scheme can significantly improve the performance of worm detection systems in terms of the tradeoff between detection rate and false positive rate.

3. Significance of Proposed Work

Our work has broad impacts on Internet worm research. The significance of this dissertation research will be as follows.

1) Our Proposed Work Is Fundamental. We use analytical tools including game theory, pattern recognition, and information theory to carry out a thorough study on approaches of countermeasures. For example, using game theory, we systematically model the interactions between the attacker and defender and consequently derive analytical results. In particular, through the process, we see that an integration of multiple defensive schemes (e.g., detection and forensics analysis) is critical for defending against worms that manipulate their propagation traffic in a smart manner. Using information theory, we map the attacks that perform probing traffic to identify location infrastracture of defense system to coding schemes for communication channels, thereby developing countermeasures that enable control on the traffic-rate change of monitors and derive theoretical bounds on the amount of time required by attack regardless of the specific attacking strategies (i.e., coding schemes) taken by the attackers.

2) Our Proposed Work Is Practical. Our techniques developed for countermeasures are compatible with the existing Internet worm defense infrastructure and hence can be used for real-world systems. In particular, since our work also uses a large number of real-world worm executables to carry out experiments, our proposed countermeasure for detecting the dynamic signature of worm program execution can be easily used by a real-world system. In addition, since our work uses traffic data provided by the Internet Threat Monitoring (ITM) system, a well deployed Internet worm defense system, our proposed countermeasures for detecting features of worm related traffic can be easily used by a real-world system.

3) Our Proposed Work Is Extensible. We develop a framework that allows us to study both traffic related features and non-traffic related features, thereby allowing us to develop

countermeasures against worm attacks. There are a number of possibilities for extending this research beyond this dissertation. In particular, since future worms can become more sophisticated and intelligent, our developed framework can be used by researchers to understand key features of other forms of new worm attacks and develop countermeasures against them.

4. Organization of This Dissertation

The rest of this dissertation is organized as follows: In Chapter II, we review the worm attacks and countermeasures. We first present our investigation on the traffic-based countermeasure in Chapters III and IV, then we discuss non-traffic based countermeasures in Chapters V and VI. Specifically, in Chapter III, we consider the worm attacks that use the feedback loop-control mechanisms to manipulate the propagation traffic rate and develop the countermeasure based on the feature of propagation traffic in spectrum-domain. In Chapter IV, we consider worm attacks that perform probing traffic in a stealthy manner, i.e., modulated by PN-code, to obtain the location infrastructure of the defense system and develop countermeasures based on monitoring the traffic-rate change of each monitor in a time-series manner. In Chapter V, we present the countermeasure based on dynamic signature of program execution which can effectively defend against new unseen worm attacks. In Chapter VI, we present the countermeasures based on attackers' conflicted-objectives against worm attacks that can dynamically manipulate their patterns. In Chapter VII, we present the countermeasure based on incorporating the defender's reputation that sacrifices its performance in the first few rounds to establish a reputation of toughness, in return for much higher payoff in the long run. Finally, we conclude this dissertation research with a brief summary in Chapter VIII.

CHAPTER II

REVIEWS OF WORM ATTACKS AND COUNTERMEASURES

In this chapter, we first briefly review the worm attacks, and then review countermeasures against worm attacks.

1. Worm Attacks

Generally speaking, the simple model of worm attack is described as follows: a worm demonstrates behavior similar to that of biological viruses, in terms of their self-propagating nature. Specifically, a worm attack usually begins when the worm attacker (or propagator) identifies vulnerable computers on the Internet, exploiting their vulnerabilities to obtain access to them, and then infecting (i.e., uploading the worm) them. Once a computer is infected, the attack becomes "automatic": A worm from the infected computer will recursively identify other vulnerable computers and try to infect those as well. In this way, the worm propagates itself to other computers on the Internet. From this simple model, we see that a worm attack will generate propagation traffic (i.e., messages that intend to identify vulnerable computers).

Since worm attacks have always posed very dangerous threats to the Internet, much effort has gone into studying, analyzing, and modeling the propagation behavior of worm attacks. For example, Kephat *et al*. in [16, 17] conducted early work on modeling a computer virus based on the epidemiology model. Staniford *et al*. in [18] studied various worms and modeled their propagation. Chen *et al*. in [10] analyzed the propagation of worms based on a discrete time model. Zou *et al*. in [11] analyzed the propagation of a worm under dynamic quarantine defense. Moore *et al*. in [2] modeled and analyzed "Slammer" worm. Zou *et al*. in [19] modeled "Code-Red" worm. Adversely, the worm attacks such as self-adaptive worms studied in this dissertation generalize worms that deliberately manipulate the propagation traffic and reduce the probability

of detection. The self-adaptive worms share some similarity in spirit with polymorphic worms that manipulate the byte stream of worm payload in order to reduce the probability of detection by payload signature-based detection [20]. All of these worms belong to the simple attack model that only generates propagation traffic (i.e., messages that intend to identify vulnerable computers).

With defensive systems in place nowadays, worms have correspondingly become more sophisticated than the simple example mentioned above. In particular, from the site of defense, Internet Threat Monitoring (ITM) systems have now been developed and deployed [21, 22], since CAIDA began to implement the network telescope to monitor Internet traffic in 2001 [23]. This kind of system is well adopted and similar to other existing worm detection systems such as the Cyber center for disease controller [18], Internet motion sensor [24], SANs ISC (Internet Storm Center) [25], Internet sink [21], network telescope [22], and CAIDA [26]. An ITM system usually consists of a number of monitors and a data center. Each monitor of an ITM system is responsible for monitoring traffic targeted to a range of IP addresses and periodically reports the collected traffic logs to the data center. The data center analyzes the traffic logs and posts summarized reports for alarming Internet worm attacks, which are usually publicly accessible. To better defeat this system via hiding itself, instead of launching the attack directly, the worm attacker uses probing messages to locate the monitors, bypassing them and reducing the probability of detection. Consequently, this kind of sophisticated worm not only propagates traffic, but it generates probing traffic as well.

For worms using the sophisticated attack model to better defeat defense systems via hiding itself, Bethencourt and Shinoda *et al*. in [27, 28] studied that ITM systems can be exploited by probing attack to locate monitors. Their techniques of locating monitors require a high volume probing traffic to be generated. This visible high traffic volume also increases the probability of

detection. Conversely, the low-rate probing attack studied in this dissertation focuses on the probing traffic in a stealthy manner, based on the Direct Sequence Spread Spectrum (DSSS) technique, utilizing a Pseudo-Noise (PN) code. This work is also closely related to other research efforts in network security. Kohno *et al*. in [29] presented a technique of sending messages to remotely fingerprint computers, exploiting small, microscope clock deviations in computers.

2. Countermeasures

In order to counteract worm attacks, there are two important steps that the defender needs to perform: worm detection and post-detection migration. Worm detection aims to identify worm propagation on the Internet. Once a worm is detected, the post-detection migration techniques can be deployed to slow down and even stop worm propagation. Some commonly adopted migration strategies include blocking/filtering propagation traffic and immunizing vulnerable computers (e.g., by releasing patches to the vulnerabilities) [10, 11, 12, 13, 14, 15]. In this dissertation, we focus on countermeasures based on worm detection as the first-line worm defense. As we mentioned in Chapter I, such countermeasures can be generally classified into two classes, known as the traffic-based countermeasure and non-traffic based countermeasure. In the following, we will overview the countermeasures related to these classes.

2.a. Traffic-Based Countermeasure

Recall that traffic-based countermeasures are those that detect worm attacks by monitoring, collecting, and analyzing the traffic generated by worm attacks. From the defense perspective, since the worm attack generates two types of traffic (propagation and probing traffic) as described previously, a defender may monitor, collect, and analyze these two types of traffic and hence detect worm attacks, via identifying traffic-related features. In order to develop these

kinds of countermeasures, it is critical to identify types of traffic generated by worm attacks and understand their characteristics.

Recall that a worm attack will generate propagation traffic (i.e., messages that intend to identify vulnerable computers). For the countermeasures based on features of propagation traffic, many detection schemes have been proposed [30, 31, 32, 33]. There are some schemes based on the observation that propagation traffic displays easily identifiable patterns, e.g., high volume, large variance, and exponentially increasing trends, etc. Generally, for these types of countermeasures, there are two types of schemes: threshold-based and trend-based detection. As examples of threshold-based detection, Venkataraman and Weaver *et al*. in [31] studied the scheme of using the mean value of traffic volume to determine the worm propagation. Wu *et al.* in [32] studied the scheme of using the variance of traffic volume to determine the worm propagation. As an example of trend-based detection, Zou *et al*. in [30] studied the scheme of using the exponential increase trend of traffic volume to determine the worm propagation. There are also other schemes that are based on destination distribution of propagation traffic. For example, Lakhina *et al*. in [33] studied the scheme of using traffic distribution (summarized by entropy) to classify various anomalies, including distribution of destination IP address to classify various anomalies. Lim *et al*. in [34, 35] also considered the header of destination IP addresses and adopted video and image processing based techniques, such as “scene change analysis” to reveal sudden changes in traffic anomalies. Conversely, in this dissertation, we investigate a new detection scheme that identifies the propagation traffic feature in the frequency domain and is able to detect worm attacks that adopt the feedback loop-control mechanism to manipulate their propagation traffiic and cause behavior similar to the background non-worm traffic.

Several studies of worm attacks and their countermeasures have also been carried out based on features of probing traffic. For example, Bethencourt and Shinoda *et al*. in [27, 28] studied an

attack scheme to locate the monitors of ITM systems. To the best of our knowledge, little work has been performed beyond very basic discussion in [27, 28]. In this dissertation, we will not only consider worm attacks that directly probe target networks, but we will also study those that perform probing in a stealthy manner and develop countermeasures against such attacks.

2.b. Non-Traffic Based Countermeasure

The traffic-based countermeasures are simple, efficient and easy to implement. Nevertheless, these detection schemes have limitations and cannot provide a complete solution for defending against worm attacks. On one hand, it is hard to use the traffic-based countermeasure to detect worms that spread via E-mail systems, instant messenger, or peer-to-peer applications, since their traffic is difficult for ITM systems to observe. On the other hand, worm attacks may have full control of traffic. Thus, traffic-based countermeasures must consequently adapt themselves in order to be effective.

As supplementary approaches against worm attacks, non-traffic based countermeasures are those that detect worm attacks without being limited to monitoring, collecting, and analyzing the traffic generated by worm attacks. In order to develop these kinds of countermeasures, it is critical to identify what types of non-traffic related features must be generated by worm attacks or effectively to worm attacks. Then we can have better understanding of their features and develop countermeasures.

For the non-traffic based countermeasures, many existing schemes have been proposed to detect the signature of worm executables [20, 36, 37, 38]. Specifically, there are some research efforts that focus on examining constant byte steams as signature in the worm program [20, 38, 39, 40], such as the list of Dynamic Link Libraries (DLLs), functions and specific ASCII strings extracted from the executable headers. There is additional research focusing on program models.

For example, Feng *et al*. [40] proposed a formal analysis framework for pushdown automata (PDA) models. Based on this framework, they studied program analysis techniques, incorporating system calls or stack activities. Wagner *et al*. in [41] proposed an approach that analyzes program executables and generates a non-deterministic finite automaton (NDFA) or a non-deterministic pushdown automaton (NDPDA) from the global control-flow graph of the program. The automaton was then used to monitor the program execution on-line. Gao *et al*. in [42] presented an approach for detecting anomalous behavior of an executing process. The basic idea of their approach is that processes potentially running the same executable should behave similarly in response to a common input.

These approaches are capable of identifying non-traffic based features generated by worm attacks and can be used to detect worm attacks. However, if new unseen worms appear in the future and a worm becomes smarter to manipulate these features, the effectiveness of these schemes will be significantly reduced. In order to address this problem, the defender needs to focus on the comparatively invariant perspectives of worm attacks. Particularly, in this dissertation we consider the following three approaches. First, we will develop one novel non-traffic countermeasure which aims to detect new unseen worms including “polymorphic” worms that have unseen signatures or change their signatures during propagation. Second, we note that no matter how a worm attacker changes its strategies, one thing it cannot change is its objectives. To this end, we develop one novel non-traffic based countermeasure by testing an important non-traffic feature − contradicted objectives of worm attacks. Third, we consider real-world system settings with multiple incoming worm attackers that collaborate by sharing the history of their interactions with the defender and we propose a generic reputation-aware countermeasure scheme to improve the performance of worm detection by incorporating the defender’s reputation.

CHAPTER III

COUNTERMEASURE BASED ON PROPAGATION TRAFFIC

In the following two chapters, we will develop traffic-based countermeasures against different worm attacks. In this chapter, we focus on developing the countermeasure based on propagation traffic.

1. Overview

In this chapter, we consider a new class of worms referred to as camouflaging worm (C-Worm in short). The C-Worm has a self-propagating behavior similar to traditional worms, i.e., it intends to rapidly infect as many vulnerable computers as possible. However, the C-Worm is quite different from traditional worms in a way that it camouflages any noticeable trends of its propagation traffic over time. Specifically, the camouflage is achieved by manipulating the propagation traffic volume launched by worm infected computers. Such a manipulation of the propagation traffic volume prevents exhibition of any exponentially increasing trends or even crossing of thresholds that are tracked by existing traffic volume-based detection schemes [30, 31, 32].

In order to detect such worm attacks, we comprehensively analyze C-Worm propagation traffic in both the time and frequency domains. We observe that although the C-Worm propagation traffic shows no noticeable trends in the time domain, it demonstrates a distinct pattern in the frequency domain. Specifically, there is an obvious concentration within a narrow range of frequencies. This concentration is inevitable since the C-Worm adapts to the dynamics of the Internet in a recurring manner for manipulating and controlling its overall propagation traffic volume. The above recurring manipulations involve steady increase followed by a decrease in the propagation traffic volume, such that the changes do not manifest as any trends in

the time domain or such that the propagation traffic volume does not cross thresholds that could reveal the C-Worm propagation.

In the following, we first introduce the C-Worm and then present the countermeasure based on the feature exposed in the spectrum domain of propagation traffic.

2. C-Worm Propagation

2.a. Overview

For the C-Worm, the simplest way to manipulate propgation traffic volume is to randomly change the number of worm instances conducting port-scans. However, this method may not be able to circumvent the detection. The reason is that the overall propagation traffic volume still shows an increasing trend with the progress of worm propagation and as more and more computers are being infected, they, in turn, take part in scanning other computers. As a result, the C-Worm may introduce a feed-back loop control for regulating its propagation speed according to the propagation status. As we mentioned earlier, in order to effectively circumvent the detection, the propagation traffic for the C-Worm should be comparatively slow and variant enough to not show any notable increasing trends over time. Note that a very slow propagation of the C-Worm is also not desirable, since it delays rapid infection damage to the Internet. Hence, the C-Worm needs to adjust its propagation so that it is neither too fast to be easily detected, nor too slow to delay rapid damage on the Internet.

To regulate the C-Worm propagation traffic volume, we introduce a loop-control parameter called attack probability $p(t)$ for each worm infected computer. $p(t)$ is the probability that a C-Worm instance participates in the worm propagation (i.e., scans and infects other computers) at time t. For the C-Worm, $p(t)$ need not be a constant value and can be set as a time varying function.

In order to achieve the camouflaging behavior, the C-Worm needs to obtain an appropriate $p(t)$ to manipulate its propagation traffic. Specifically, the C-Worm will regulate its overall propagation traffic volume such that: (i) it is similar to non-worm scan traffic in terms of the traffic volume over time, (ii) its does not exhibit any notable trends such as an exponentially increasing pattern or any mono-increasing pattern even when the number of infected computers increases over time, and (iii) the average volume value of the overall traffic is sufficient to make the C-Worm propagate fast enough to cause rapid damage on the Internet.

We assume that a worm attacker intending to manipulate propagation traffic volume follows a random distribution with mean M_C^*. This M_C^* can be regulated in a random fashion during the worm propagation in order to camouflage the propagation of C-Worm. Correspondingly, the worm instances need to readjust their attack probability $p(t)$ in order to ensure that the total number of worm instances that launch the scans is approximately M_C^*.

To regulate M_C^*, it is obvious that $p(t)$ has to be decreased over time since $M(t)$ keeps increasing during worm propagation. We can determine $p(t)$ using a simple function as follows: $p(t) = M_C^*/\hat{M}(t)$, where $\hat{M}(t)$ represents the estimation of $M(t)$ at time t. From the above expression, we know that the C-Worm needs to obtain the value of $\hat{M}(t)$ (as close to $M(t)$ as possible) in order to generate an effective $p(t)$. Here, we discuss one approach for the C-Worm to estimate $M(t)$. The basic idea is as follows: A C-Worm could estimate the percentage of computers that have already been infected over the total number of IP addresses as well as $M(t)$, through checking a propagation attempt as a new hit (i.e., hitting an uninfected vulnerable computer) or a duplicate hit (i.e., hitting an already infected vulnerable computer). This method requires each worm instance (i.e., infected computer) to be marked by a watermark which indicates that this computer has been infected. Thus, when a worm instance (for example, computer A) scans one infected computer (for example, computer B), then computer A will

detect such a watermark, thereby becoming aware that host B has been infected. Through validating such watermarks during the propagation, a C-Worm infected computer can estimate $M(t)$. This method is similar to that used by the "self-stopping" worm discussed in [43]. There are other approaches to achieve this goal, such as incorporating the Peer-to-Peer techniques to disseminate information through secured IRC channels [44, 45].

2.b. Effectiveness

We now demonstrate the effectiveness of C-Worm in evading worm detection through controlling $p(t)$. In this context, we use two metrics to assess a detection scheme. One is the *Detection Time* (DT) and the other is the *Maximal Infection Rate* (MIR). These two metrics are used to measure the effectiveness of the worm attacks in the presence of worm defense systems. Detection time quantifies the detection speed of the detection scheme and maximal infection rate quantifies the damage caused by a worm before being detected. The purpose of any detection scheme is to rapidly minimize the damage caused by a worm. Hence, these two metrics can be used to quantify the effectiveness of any worm countermeasure. As the values increase, the worm attack performance improves and the detection performance worsens.

Given random selection of M_C^*, we generate three C-Worm attacks (viz., C-Worm 1, C-Worm 2 and C-Worm 3) that are characterized by different selections of mean and variance magnitudes for M_C^*. In our simulations, we assume that the scan rate of traditional pure random scan (PRS) worm follows a normal distribution $S_n = N(40, 40)$ (note that if the scan rate generated by above distribution is less than 0 , we set the scan rate as 0). We also set the total number of vulnerable computers on the Internet as 360,000 which is the total number of infected computers in "Code-Red" worm incident [1].

Table III-1. Maximal Infection Rate (MIR) for Existing Traffic Volume-Based Detection Schemes

Detection Schemes	PRS worm	C-Worm 1	C-Worm 2	C-Worm 3
Mean	4.8%	100%	100%	28%
VAR	5.0%	100%	100%	100%
TREND	3.1%	100%	100%	100%

Table III-2. Detection Time (DT) for Existing Traffic Volume-Based Detection Schemes

Detection Schemes	PRS worm	C-Worm 1	C-Worm 2	C-Worm 3
Mean	2290	Inf	Inf	4803
VAR	2340	Inf	Inf	Inf
TREND	2134	Inf	Inf	Inf

Table III-1 and Table III-2 show how the C-Worm is able to effectively defeat the existing traffic volume-based detection schemes. The data of these two tables show the detection results of three representative traffic volume-based detection schemes (denoted by MEAN [31], VAR [32], and TREND [30]) on PRS worms and different C-Worms. For fairness, we set the parameters for these three detection schemes, so that all schemes can achieve similar low false positive rates, i.e., less than 1%. Remark that the false positive rate is the probability that a detection system detects the existence of worm propagation when there is actually no occurrence of worm

propagation. Although all three schemes are effective while detecting PRS worm attacks, they fail in detecting the C-Worm attacks. For example, all the schemes completely fail to detect the C-Worm 1 and 2. Only MEAN can detect the C-Worm 3, but only after a considerably large detection time of 4803 minutes and an unimpressive maximal infection rate of 28%.

2.c. Discussion

Although in this chapter we only demonstrate effectiveness of C-Worms against existing traffic volume-based detection schemes, the design principle of C-Worm can be extended to defeat other newly developed detection schemes, such as destination distribution-based detection [33, 34, 35]. In the following, we discuss the preliminary idea.

Recall that the attack target distribution-based schemes intend to analyze the distribution of attack targets (the scanned destination IP addresses) as basic detection data to capture the fundamental feature of worm propagation, i.e., continuously scanning different targets, which is not expected in non-worm scan traffic. However, our initial investigation shows that the worm attacker is still able to defeat such a countermeasure via manipulating the attack target distribution. For example, the attacker may launch a portion of scan traffic bound for some IP addresses monitored by ITM system. Recall that those dedicated IP addresses monitored by ITM system can be obtained by launching probing attacks or via other means, which will be studied in Chapter IV.

Using port 135 reported by SANs ISC as an example, we analyze the traces and obtain the traffic target distribution in a window lasting for 10 mins. Following existing work [33], we use entropy as the metrics to measure the attack target distribution. Fig. III-1 shows the Probability Density Function (PDF) of background traffic's entropy values. We also simulate the worm propagation traffic which allocate a portion of scan traffic bound for IP addresses monitored by

the ITM system, then we obtain the PDF of entropy value for combined traffic including both worm propagation and background traffic. From Fig. III-1, we know that when the attacker uses a portion of attack traffic to manipulate the target distribution, the entropy-based detection scheme can be degraded significantly. For example, when the attacker uses 10% traffic to manipulate the traffic's entropy value, the false positive rate of entropy-based detection scheme is 14%. When the attacker uses 30% traffic to manipulate the traffic's entropy value, the false positive rate becomes 40%. Hence, in order to preserve the performance, entropy-based detection scheme needs to evolve correspondingly and integrate with other detection schemes. We will perform a more detailed study of this aspect in our future work.

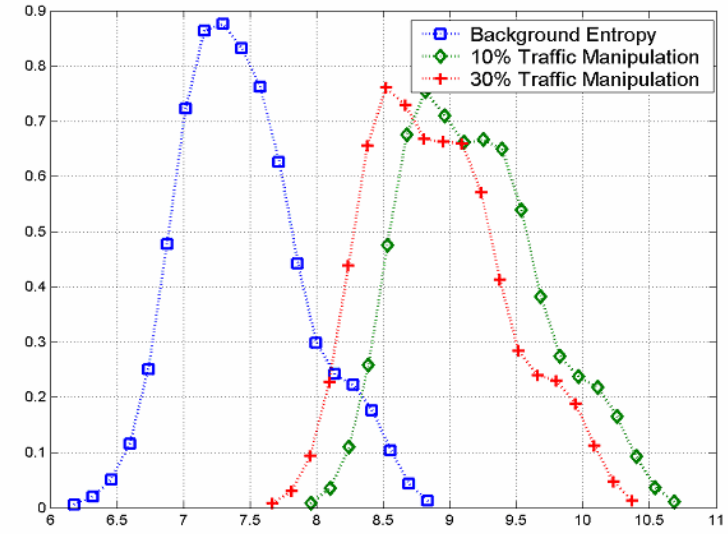


Fig. III-1. Manipulation of Attack Target Distribution Entropy

3. Detection of the C-Worm

3.a. Design Rationale

In this section, we develop a novel spectrum-based detection scheme. Recall that the C-Worm goes undetected by detection schemes that try to determine the worm propagation volume only in the time domain. Our detection scheme captures the distinct pattern of the C-Worm in the frequency domain, and thereby has the potential of effectively detecting the C-Worm propagation.

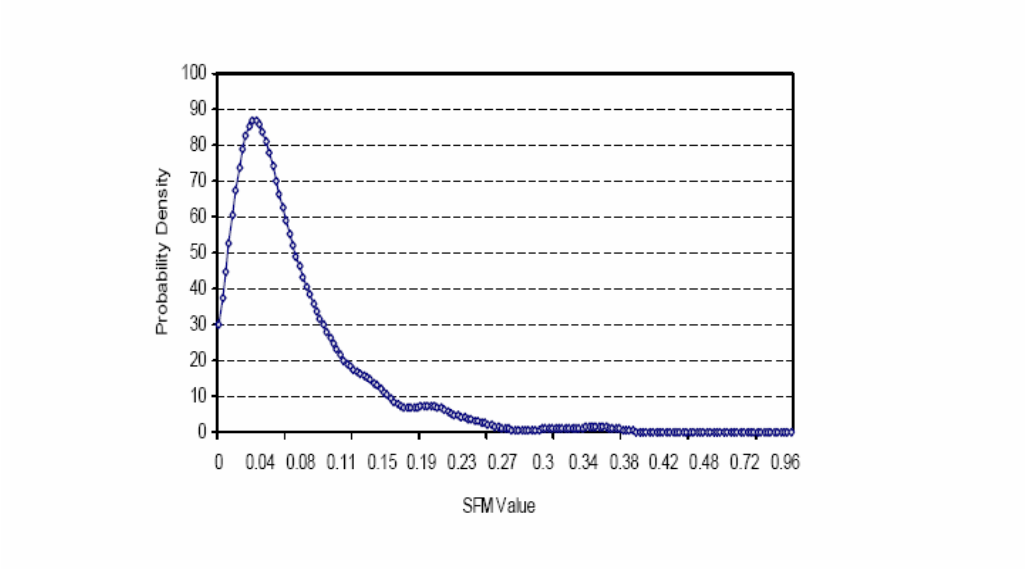


Fig. III-2. PDF of SFM on C-Worm Traffic

In order to identify the C-Worm propagation in the frequency domain, we use the distribution of *Power Spectral Density* (PSD) and its corresponding *Spectral Flatness Measure* (SFM) of the propagation traffic. Particularly, PSD describes how the power of a time series is distributed in the frequency domain. Mathematically, it is defined as the *Fourier transform* of the auto-correlation of a time series. In our case, the time series corresponds to the changes in the number

of worm instances that actively conduct the propagation over time. The SFM of PSD is defined as the ratio of geometric mean to arithmetic mean of the coefficients of PSD. The range of SFM values is [0, 1] and a larger SFM value implies flatter PSD distribution and vice versa.

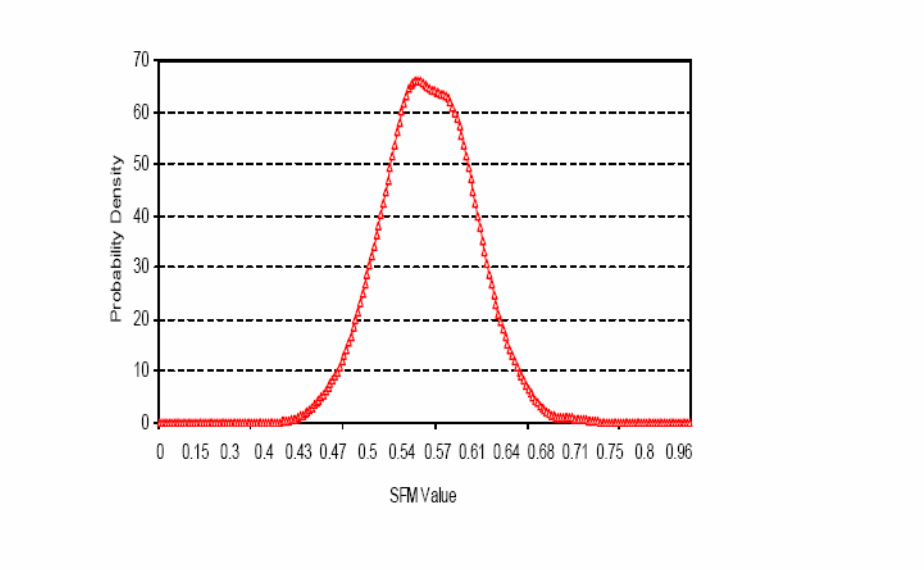


Fig. III-3. PDF of SFM on Non-Worm Traffic

To illustrate SFM values of both the C-Worm propagation and normal non-worm scan traffic, we plot the Probability Density Function (PDF) of SFM for both C-Worm propagation and normal non-worm scan traffic as shown in Fig. III-2 and Fig. III-3, respectively. Note that we only show the data for port 8080 as an example, and other ports show similar observations. From this figure, we know that the SFM value for normal non-worm scan traffic is very large (e.g., SFM in [0.5, 0.6] has much higher density compared with other magnitudes). The C-Worm data shown in Fig. III-2 is based on 800 C-Worm attacks generated by varying attack parameters defined in Section 2, such as $p(t)$ and M_C^*. From this figure, we know that the SFM value of the C-Worm attacks is much smaller (e.g., SFM in [0.04, 0.1] has high density). From the above two

figures, we can observe that there is a clear demarcation range of SFM in (0.3, 0.38) between the C-Worm propagation and normal non-worm scan traffic. As such, the SFM can be used to effectively detect the C-Worm propagation traffic.

The large SFM values of normal non-worm scan traffic can be explained as follows. The normal non-worm scan traffic does not tend to concentrate at any particular frequency since its random dynamics is not caused by any recurring phenomenon. The small value of SFM can be reasoned by the fact that the frequency of C-Worm propagation traffic is within a narrow-band. Such concentration within a narrow range of frequencies is unavoidable since the C-Worm adapts to the dynamics of the Internet in a recurring manner for manipulating the overall propagation traffic volume. In reality, the above recurring manipulations involve steady increase followed by a decrease in the propagation traffic volume.

3.b. Spectrum-Based Detection Scheme

We now present the details of our spectrum-based detection scheme. Similar to other detection schemes [30, 32], we use a "destination count" as the number of the unique destination IP addresses targeted by launched scans during worm propagation. To understand how the source count data is obtained, we recall that an ITM system collects logs from distributed monitors across the Internet. With reports in a sampling window W_s, the destination count $X(t)$ is obtained by counting the unique destination IP addresses in received logs.

To conduct spectrum analysis, we consider a sliding window W_d in the worm detection system. W_d consists of q continuous detection sampling windows and each sampling window lasts W_s. The detection sampling window is the unit time interval to sample the detection data (e.g., the destination count). Hence, at time i, within a sliding window W_d, there are q samples

denoted by ($X(i\text{-}q\text{-}1)$, $X(i\text{-}q\text{-}2)$,…, $X(i)$), where $X(i\text{-}j\text{-}1)$ (j in (1, q)) is j-th destination count from time $i\text{-}j\text{-}1$ to $i\text{-}j$.

In our spectrum-based detection scheme, the distribution of PSD and its corresponding SFM are used to distinguish the C-Worm propagation traffic from the non-worm scan traffic. In our worm detection scheme, the detection data (e.g., destination counter), is further processed in order to obtain its PSD and SFM. In the following, we detail how the PSD and SFM are determined during the processing of the detection data.

1) Power Spectral Density (PSD)

To obtain the PSD distribution for worm detection data, we need to transform data from the time domain into the frequency domain. To do so, we use a random process $X(t)$, t in [0, n] to represent the worm detection data. Assuming $X(t)$ is the destination count in time period [t-1, t] (t in [1, n]), we define the auto-correlation of X(t) by $R_X(L) = E\,[X(t)X(t+L)]$, where $R_X(L)$ is the correlation of worm detection data in an interval L. If a recurring behavior exists, a Fourier transform of the auto-correlation function of $R_X(L)$ can reveal such behavior. Thus, the PSD function (also represented by $S_X(f)$; where f refers to frequency) of the scan traffic data is determined using the Discrete Fourier Transform (DFT) of its auto-correlation function as follows,

$$\varphi(R_X(L), K) = \sum_{n=0}^{N-1} (R_X(L)) \cdot e^{-j2\pi kn/N}, \qquad \text{(III-5)}$$

where K=0, 1, …, N-1. As the PSD inherently captures any recurring pattern in the frequency domain, the PSD function shows a comparatively even distribution across a wide spectrum range for the normal non-worm scan traffic. Whereas, the PSD of C-Worm propagation traffic shows spikes or noticeably higher concentrations at a certain range of the spectrum range.

2) Spectral Flatness Measure (SFM)

We measure the flatness of PSD to distinguish the propagation traffic of the C-Worm from the normal non-worm scan traffic. To this end, we introduce the *Spectral Flatness Measure* (SFM). The SFM is defined as the ratio of the geometric mean to the arithmetic mean of the PSD coefficients [47, 48]. It can be expressed as,

$$SFM = \frac{[\prod_{k=1}^{N} S(f_k)]^{\frac{1}{N}}}{\frac{1}{N}\sum_{K=1}^{N} S(f_k)}, \quad \text{(III-6)}$$

where $S(f_k)$ is the k-th PSD coefficient for the PSD obtained from the results in (III-5). SFM is a widely existing measure for discriminating frequencies in various applications such as voiced frame detection in speech recognition [48, 49]. In general, small values of SFM imply the concentration of data at narrow frequency spectrum ranges.

Table III-3 shows the mean value of SFM based on extensive analysis of non-worm traffic data for some popular ports collected by SANs ISC. Overall, we note that the PSD distribution of non-worm scan traffic is relatively flat, thereby resulting in relatively larger magnitudes of SFM values. The above observation can be reasoned due to the fact that normal non-worm scan traffic does not tend to concentrate at any particular frequency since its random dynamics is not caused by any repeating phenomenon. Differently, the C-Worm has unpreventable recurring behavior in its propagation traffic; consequently its SFM values are comparatively smaller than the SFM values of normal non-worm scan traffic.

Table III-3. SFM Mean Value for Normal Non-Worm Scan Traffic

Port	23	25	53	113	139	445	1025	4672	6446	6881	8080	27015
SFM	0.71	0.71	0.95	0.86	0.64	0.67	0.46	0.47	0.45	0.74	0.56	0.65

3) Detection Decision Rule

We now describe the method of applying an appropriate detection rule to detect C-Worm propagation. As the SFM value can be used to sensitively distinguish the C-Worm propagation and normal non-worm scan traffic, the worm detection is performed by comparing the SFM with a predefined threshold. If the *SFM* value is smaller than a predefined threshold, then a C-Worm propagation alert is generated. The value of the threshold used by the C-Worm detection can be set based on the knowledge of statistical distribution of SFM values that correspond to the non-worm scan traffic. If we can obtain the distribution of SFM values for the C-Worm through comprehensive simulations and even real-world profiled data in the future, the optimal threshold can be obtained by applying the Bayes classification [50]. If the distribution of *SFM* values for the C-Worm is not available, based on the distribution of SFM values of the normal non-worm scan traffic, we can set an appropriate value. For example, the value can be determined by the Chebyshev inequality [50] in order to obtain a reasonable false positive rate for worm detection.

In addition, our spectrum-based scheme is also generic for detecting the PRS worms. This is due to the fact that propagation traffic of PRS worms has a constantly rapid, exponential increase. Thus, in the propagation traffic of PRS worms, the PSD values in the low frequency range are much higher compared with other frequency ranges.

3.c. Analysis

We now present a formal analysis of SFM for the C-Worm. Let the observed traffic on the countermeasure system be $Z_1 = X_1 + Y_1$, where X_1 is the random variable representing the C-Worm propagation traffic (e.g., volume, source counter) in one sampling window and Y_1 is the random variable representing the background scan traffic (e.g., volume, source counter) in one sampling window. We define $X = X_1 - \mathrm{E}[X_1]$, where $\mathrm{E}[X_1]$ is the mean value of X_1 and $Y = Y -$

E[Y_1], where E[Y_1] is the mean value of Y_1. Thus, we have $Z = X + Y$, where X and Y are independent zero-mean random variables. We assume that Z's spectrum is within the $-W \le f \le W$ range.

Based on the observations shown in Section 3.a, we approximately represent $Y_1(t)$ by white Gaussian noise, which is widely used in modeling wide-band noise in communication systems. Thus, Y can be approximately represented by a Gaussian white noise with zero mean and a variance of σ. Thus, in the total frequency band limited within the range [$-W \le f \le W$], the PSD of Y is $S_Y(f) = \sigma$, which shows that Y has a constant power spectrum and each frequency has the average power value σ.

Considering the fact that C-Worm instances adopt the control mechanism strategy to manipulate the overall propagation traffic volume, we explained how a distinct trend can be noticed in the spectrum domain, i.e., the trend being a concentration within a narrow range of frequencies on the propagation traffic of the C-Worm. Assume that the frequency of C-Worm propagation traffic counter is referred to as m (denoted by f_k), where $k = 1, \ldots, m$ and $m < W$ in the total (narrow-band) frequency range. Without loss of generality, $X(t)$ is approximately represented by

$$X(t) = \sum_{k=1}^{2m} a_k \cos(2\pi f_k t + \theta), \qquad \text{(III-7)}$$

where θ is uniformly distributed in the interval [0, 2π]) and a_k is uniformly distributed in the interval [−l, l]. Based on the relationship among autocorrelation, mean and autoconvariance, we have $R_X(\tau) = C_X(t_1, t_2) + \mathrm{E}[X(t_1)]\mathrm{E}[X(t_2)]$, where $\tau = t_2 - t_1$, $\mathrm{E}[X(t_1)] = \mathrm{E}[X(t_2)] = 0$, and $C_X(t_1, t_2) = \mathrm{E}[(X(t_1) - \mathrm{E}(X(t_1))(X(t_2) - \mathrm{E}(X(t_2))]$ is the autocovariance of a random process $X(t)$. Thus, it is easy to verify that

$$R_X(\tau) = \sum_{k=1}^{m} [\frac{a_k^2}{2} \cos(2\pi f_k \tau)]. \quad \text{(III-8)}$$

Thus, the PSD of $X(t)$ can be represented by

$$S_X(f) = \sum_{k=1}^{k=m} [\frac{a_k^2}{4} \delta(f - f_k) + \frac{a_k^2}{4} \delta(f + f_k)]. \quad \text{(III-9)}$$

As $X(t)$ and $Y(t)$ are independent random process ($S_Y(f) = \sigma$), we have

$$S_X(f) = \sum_{k=1}^{k=m} [\frac{a_k^2}{4} \delta(f - f_k) + \frac{a_k^2}{4} \delta(f + f_k)] + \sigma. \quad \text{(III-10)}$$

Define $R=a_k\delta(f_k)/4\sigma$. The SFM of $Z(t)$ can be represented by

$$S_Z(f) = \frac{\sigma^{2W-2m} R\sigma^{\frac{2m}{2W}}}{\frac{1}{2W}[2m\sigma R + \sigma(2W - 2m)]} = \frac{R^{\frac{m}{W}}}{\frac{m}{W}(R-1)+1}. \quad \text{(III-11)}$$

We can rewrite $S_Z(f)$ in (III-11) as the function of R as

$$F(x) = \frac{x^t}{t(x-1)+1}, \quad \text{(III-12)}$$

where $x = R$, $t = m/W < 1$. As

$$F'(x) = \frac{tx^{t-1}(x-1)(t-1)}{[t(x-1)+1]^2} < 0, \quad \text{(III-13)}$$

the function $S_Z(f)$ is a decreasing function of x (= R) and it is observable that

$$R = \frac{a_k 4\delta(f)}{\sigma} + 1 >> 1, \quad \text{(III-14)}$$

(due to the Dirac's δ function property), $S_Z(f) \rightarrow 0$. Thus, the SFM of C-Worm is close to 0.

4. Performance Evaluation

In this section, we report our evaluation results that illustrate the effectiveness of our spectrum-based detection scheme against both the C-Worm and the PRS worm in comparison with existing representative volume-based detection schemes. In addition, we also consider the destination distribution-based detection schemes and evaluate their performance against the C-Worm.

4.a. Evaluation Methodology

1) Evaluation Metrics

In order to evaluate the performance of any given detection scheme against the C-Worm, we use the following metrics. The first two metrics are the *Detection Time* (DT) and the *Maximal Infection Rate* (MIR) defined in Section 2. Recall that detection time is defined as the time taken to successfully detect the worm attack from the moment the worm propagation starts. It quantifies the detection speed of a detection scheme. Maximal infection rate defines the ratio of an infected computer number over the total number of vulnerable computers up to the moment when the worm propagation is detected. It quantifies the damage caused by a worm before being detected. The objective of any detection scheme is to minimize the damage caused by a rapid worm propagation. Hence, MIR and DT can be used to quantify the effectiveness of any worm detection scheme. The higher the values, the more effective the worm attack and the less effective the detection. In addition, we use other two metrics called the *Detection Rate* (P_D) and *False Positive Rate* (P_F). P_D is defined as the probability that a detection scheme can correctly identify a worm attack. The P_F is defined as the probability that a detection scheme mistakenly identifies a nonexistent worm attack.

2) Evaluation Setup

In our evaluations, we set the total number of vulnerable computers on the Internet as 360,000 [1]. For the scan rate S (number of scans per minute), we choose different scan rates for infected computers (worm instances). In our evaluation, the scan rates are predetermined and follow a Gaussian distribution $S = N(S_m, S_\delta)$, where S_m and S_δ are in [20, 64], similar to those used in [30].

We simulate the C-Worm attacks by varying the attack parameters, such as control parameter $p(t)$ and the number of worm instances participating the scan M_C^* defined in Section 2. The M_C^* follows the Gaussian distribution. Particularly, its mean is randomly selected in (12000, 75000) and standard deviation is randomly selected in (0.2, 100). We simulate different C-Worm propagation traffic by varying these values. The detection sampling window W_s is set to 5 minutes and the detection sliding window W_d is set to be incremental from 80 min to 800 min. The incremental selection of W_s from a comparatively small window to a large window can adaptively reflect the worm scan traffic dynamics caused by the C-Worm propagation at various speeds. We choose the setting of the detection sampling window to be short enough in order to provide enough sampling accuracy, as prescribed by Nyquist's sampling theory. Also, we choose the detection sliding window to be long enough to capture adequate information for spectrum-based analysis [48].

In practice, since detection systems analyze port-scan traffic blended with the non-worm scan traffic, we replay the real-world traces as non-worm scan traffic (as the background noise to worm propagation traffic) in our simulations. In particular, we used real-world trace (Shield logs dataset) from 01/01/2005 to 01/15/2005 collected by a ITM system called SANs ISC. Note that SANs ISC maintained by the SANs Institute have gained popularity among the Internet security community in recent years. ISC collects firewall and Intrusion detection system logs, which indicate port-scan trends from approximately 2000 organizations that monitor up to 1 million IP

addresses. We choose the scan traffic logs for port 8080 as an example for profiling the non-worm scan traffic.

4.b. Evaluation Results of Traffic Volume-Based Detection Schemes

We evaluate our proposed spectrum-based detection scheme by comparing its performance with three existing propagation traffic volume-based detection schemes. The first scheme is the volume mean-based (MEAN) detection scheme which uses the mean value of propagation traffic to detect worm propagation [31]; the second scheme is the trend-based (TREND) detection scheme which uses the increase trend of propagation traffic volume to detect worm propagation [30]; and the third scheme is the victim number variance-based (VAR) detection scheme which uses the variance of the propagation traffic volume to detect worm propagation [32].

We define our spectrum-based detection scheme as SPEC. For the off-line training, we use 1000 worm attacks that include both the C-Worm (800 C-Worm attacks) and PRS worms (200 PRS worm attacks). For fairness, we set the detection parameters for our SPEC scheme and the other three detection schemes, so that all detection schemes achieve a similar false positive rate (P_F) below 2%.

In the following, we first evaluate the performance of our spectrum-based detection scheme for C-Worms. Following this, we evaluate the performance of our spectrum-based detection scheme for PRS worms.

1) Detection of C-Worms

Table III-4 shows the detection results of different detection schemes against the C-Worm. The results have been averaged over 500 C-Worm attacks. From this table, we can observe that existing detection schemes are not able to effectively detect the C-Worm and their detection rate. (P_D) values are significantly lower in comparison with our spectrum-based detection schemes

(SPEC). For example, SPEC achieves the detection rate of 98%, which is at least 3-4 times more accurate than detection schemes such as VAR and MEAN that achieve detection rate values of only 48% and 14%, respectively.

Our SPEC detection schemes also achieve good detection time (DT) performance in addition to the high detection rate values indicated above. In contrast, the detection time of existing detection schemes have relatively larger values. As a consequence of the detection time values, we can see that the C-Worm propagation is effectively contained by SPEC as demonstrated by the lower values of maximal infection rate (MIR) for the SPEC. Since the detection rate values for the existing detection schemes are relatively small, obtaining low values of maximal infection rate for those schemes are not as significant as those for SPEC.

Table III-4. Detection Results of Traffic Volume-Based Schemes against C-Worm

Schemes	VAR	TREND	MEAN	SPEC
Detection Rate (P_D)	48%	0	14%	98%
Maximal Infection Rate (*MIR*)	14.4%	100%	7.5%	1.1%
Detection Time (*DT*)	2567	Inf	1838	1749

2) Detection Performance for PRS Worms

We evaluate the detection performance of different detection schemes for PRS worms. The detection performance results have been averaged over 500 PRS worm attacks. We observe that our SPEC schemes achieve 100% detection rate (P_D) while detecting traditional PRS worms in comparison with existing worm detection schemes that have been specifically designed for detecting PRS worms.

In view of emphasizing the performance of our SPEC scheme with the existing worm detection schemes, we plot the maximal infection rate (MIR) and detection time (DT) results in Figs. III-4 and III-5 for different scan rates, respectively. We can observe from these figures that the maximal infection rate and detection time results of our spectrum-based scheme are comparable or even better than other existing worm detection schemes. For example, when the mean scan rate is 70/min, our SPEC scheme achieves a detection time of 1024 mins, which is faster than that of VAR and MEAN schemes with values 1239 min and 1161 min, respectively. For the same mean scan rate of 70/min, SPEC achieves a maximal infection rate of 0.03, which is comparable to TREND's MIR value and is less than 50% of the MIR value for the VAR and MEAN detection schemes. The effectiveness of our spectrum-based scheme is based on the fact that PRS worm propagation traffic shows a constantly rapid exponential increase. Thus, SFM values are relatively small due to PSD concentration at the low frequency range in the case of PRS worms.

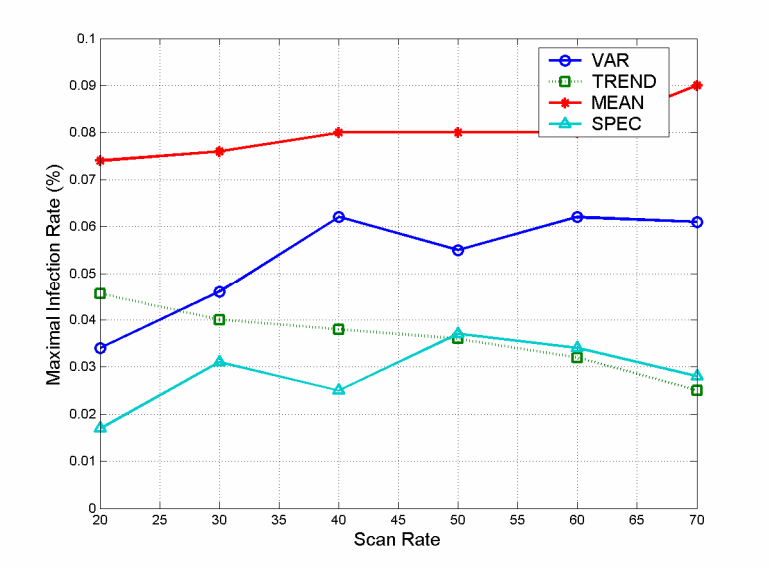


Fig. III-4. Maximal Infection Rate on PRS Worms

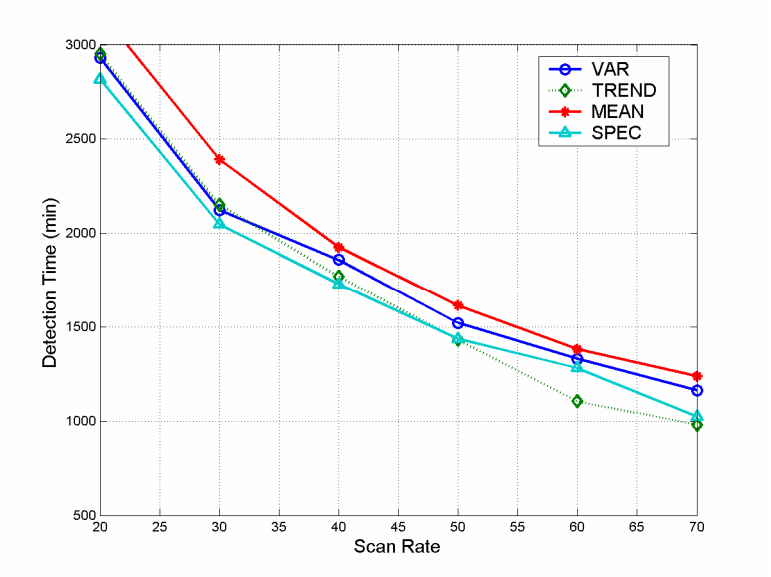


Fig. III-5. Detection Time on PRS Worms

4.c. Evaluation Results of Traffic Distribution-Based Detection Schemes

In Section 4.b, we evaluate the detection performance of our proposed scheme along with other three detection schemes. Each of these detection schemes belong to the traffic volume-based detection category due to the fact that traffic volume is used as the main detection feature. As we mentioned earlier, there are other schemes based on the destination distribution of worm propagation traffic [33, 34, 35]. Taking into consideration this category of detection schemes, we evaluate two additional schemes against the C-Worm. The first one is the entropy-based detection scheme [33] which uses entropy to measure the traffic destination distribution feature raised by worm propagation. For this detection scheme, we record all scan traffic data in each sampling window and then calculate the entropy. The sliding detection window is set to 10 (consists of 10 sample windows). If the average value of the entropy within a sliding detection window is larger than predefined threshold, which is determined based on the statistical profile

of background traffic. Other measures, such as correlation-coefficient, have also been showing the effective capture of the destination distribution characteristics raised by worm propagation. The second scheme is an extension of the first one, incorporating the wavelet analysis. Based on the time-series of data (the entropy value in each sampling window), we carry out discreet wavelet transform (DWT) and record the coefficients of wavelet analysis at different levels. In our case, the anomaly detection is based on approximate coefficients of level 4 and 5, which represent the signal anomaly in a low-frequency range. In our experiment, the length of time-series DWT is set to 50 and each data represents the entropy value in one sampling window. The wave in our experiment uses the *Daubechies* orthogonal wavelet. The parameters for generating C-Worms is the same as those shown in Section 4.a.

Table III-5. Detection Results for Target Distribution-Based Schemes against C-Worm

Schemes	Entropy	Entropy with Wavlet	SPEC
Detection Rate (P_D)	98%	99%	98%
Maximal Infection Rate (*MIR*)	0.8%	0.5%	1.1%
Detection Time (*DT*)	1649	1548	1749

Table III-5 shows the detection results of destination distribution-based detection schemes in comparison with our proposed scheme against the C-Worm. From this table, we can see that our proposed scheme achieves comparable detection performance against the C-Worm in terms of detection rate, maximal infection rate, and detection time. However, our scheme is a somewhat slower, resulting in worse detection time and maximal infection rate. This is expected and can be explained by the following two facts: First, our spectrum-based scheme heavily relies on traffic

volume. Second, the C-Worm studied in this chapter is mainly traffic volume-based. That is, the C-Worm in this chapter only limits the manipulation of traffic volume, aiming to defeat existing traffic volume-based detection schemes. However, as shown in Section 2.c, it is possible that a worm attacker can adopt other strategies (e.g., manipluation of the attack target distribution) and further defeat destination distribution-based detection schemes. For example, the attacker may launch a portion of scan traffic bound for some IP addresses monitored by ITM system. Recall that those dedicated IP addresses monitored by ITM system can be obtained by launching probing attacks or via other means, which will be studied in Chapter IV.

5. Summary

In this chapter, we studied the countermeasure based on propagation traffic to defend against a specific class of worm called the C-Worm that has the capability to camouflage its propagation traffic volume and such behavior as background traffic. Our analysis and evaluation showed that, although the C-Worm successfully camouflages its propagation in the time domain, its camouflaging nature inevitably manifests as a distinct pattern in the frequency domain. Based on such observations, we developed a novel spectrum-based detection scheme to detect the C-Worm. Specifically, our spectrum-based detection scheme used the Power Spectral Density (PSD) distribution of the C-Worm propagation traffic volume and its corresponding Spectral Flatness Measure (SFM) as the key detection feature to distinguish the C-Worm propagation traffic from the normal non-worm scan traffic. The evaluation data showed that our scheme achieved superior detection performance against the C-Worm in comparison with other propagation traffic volume-based detection schemes.

CHAPTER IV

COUNTERMEASURE BASED ON PROBING TRAFFIC

In this chapter, we focus on developing the countermeasure based on probing traffic.

1. Overview

To order to defend against worm attacks, large-scale traffic monitoring across the Internet has become necessary. Developing and deploying Internet threat monitoring (ITM) systems (or motion sensor networks) is one of the major efforts in this realm. Generally, an ITM system consists of a number of monitors and a data center. The monitors are distributed across the Internet and can be deployed at hosts, routers, and firewalls, etc. Each monitor is responsible for monitoring and collecting traffic targeting to a range of IP addresses within a sub-network. The range of IP addresses covered by a monitor is also referred to as the location of the monitor. Periodically, the monitors send traffic logs to the data center and the data center analyzes the traffic logs and issues the worm attack warnings.

However, the integrity and functionality of ITM systems largely depend on the confidentiality of the IP addresses covered by their monitors, i.e., the locations of monitors. If the locations of monitors are identified, the attacker can deliberately avoid these monitors and directly attack the uncovered IP address space. It is a known fact that the number of sub-networks covered by monitors is much smaller than the total number of sub-networks in the Internet [21, 22, 25]. In other words, the IP address space covered by monitors represents a very small portion of the entire IP address space. Hence, bypassing IP address spaces covered by monitors will significantly degrade the accuracy of the traffic data collected by the ITM system in reflecting the real situation of attack traffic. Furthermore, the attacker may also poison ITM systems by manipulating the traffic towards and captured by disclosed monitors. For example, the attacker

can launch high-rate port-scan traffic to disclosed monitors and feign a large scale worm propagation. In summary, the attacker can significantly compromise the ITM system performance if he is able to disclose the locations of monitors. It is important to have a thorough understanding of such attacks and design efficient countermeasures to defend against them.

In the following, we first investigate a category of stealthy attacks called low-rate LOCcalization (lLOC) attack, which can accurately and invisibly localize the monitors in ITM systems. We then develop countermeasures to defend against such attacks. Notice that the stealthy probing attack part in this Chapter is based on the joined work between Texas A&M University and the Ohio State University. My work focused on problem definition, literature survey, mathematical analysis, and simulations.

2. Attack Model

In this section, we will discuss the lLOC attack in detail. We will first give an overview of the lLOC attack, and then present the detailed procedures of the attack, followed by additional discussions and analytical results on its mechanisms.

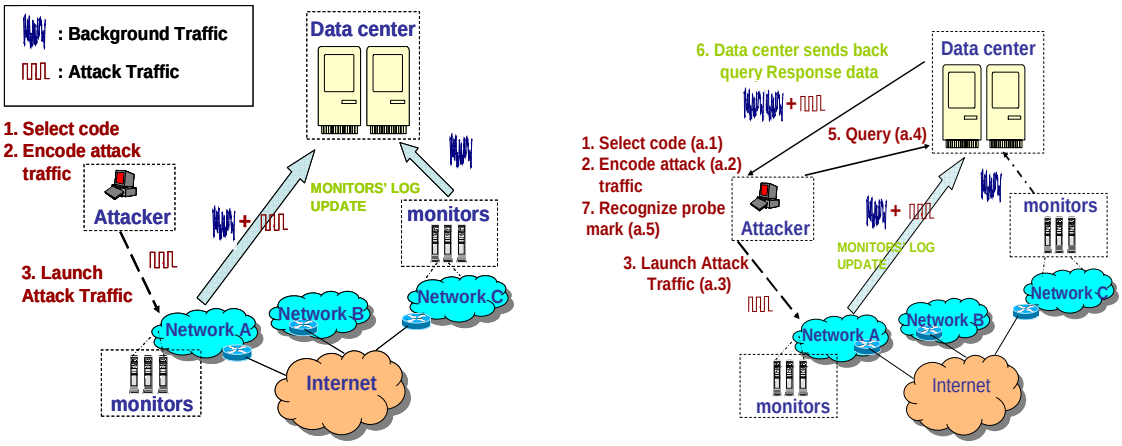
2.a. lLOC Attack

1) Workflow

Fig. IV-1 shows the basic workflow of the lLOC attack. This figure also illustrates the basic idea of the ITM system and its threats. In the ITM system, the monitors deployed at various networks record their observed port-scan traffic and continuously update their traffic logs to the data center. The data center first summarizes the volume of port-scan traffic destinated towards (and reported by) all monitors, and then publishes the report data to the public in a timely fashion.

As shown in Fig. IV-1 (a) and (b) respectively, the lLOC attack consists of the following two stages:

(a) *Attack Traffic Generation:* In this stage, as shown in Fig. IV-1 (a), the attacker first selects a code. Then, he encodes the attack traffic by *embedding* the selected code into the traffic. Lastly, the attacker launches the attack traffic towards a target network (e.g., network A in Fig. IV-1 (a)). We denote such an *embedded code pattern* in the attack traffic as the *attack mark* of the lLOC attack, and denote the attack traffic encoded as *attack mark traffic*.



(a) attack stage 1: attack traffic generation (b) attack stage 2: attack traffic decoding

Fig. IV-1. Workflow of the lLOC Attack

(b) *Attack Traffic Decoding:* In this stage, as shown in Fig. IV-1 (b), the attacker first queries the data center for the traffic report data. Such report data consist of both attack traffic and background traffic. After obtaining the report data, the attacker tries to recognize the attack mark (i.e., the code embedded in the lLOC attack traffic) by decoding the report data. If the attack mark is recognized, the report data must include the attack traffic, which means the target

network is deployed with monitors and the monitors are sending traffic reports to the ITM data center.

2) Code-Based Attack

The lLOC attack adopts a code-based approach to generate the attack traffic. Coding techniques have been widely implemented in secured communication; for example, *Morse code* is one such example. Without knowledge of Morse code, the receiver would find it impossible to interpret the carried information [51].

In the lLOC attack, we use the *pseudo-noise code* (PN-code) based attack approach, which has three advantages. First, the code is embedded in traffic and can be correctly recognized by the attacker even under the interference from background traffic, ensuring accuracy of the attack. Second, the code (of sufficient length) itself provides enough privacy. That is, the code is only known by the attacker, thereby, only the code pattern embedded in attack traffic can be recognized by the attacker. Furthermore, the code is able to carry information. A longer code is more immune to interference, and requires comparatively lower-rate attack traffic as the carrier, which is harder to be detected. All these characteristics help to achieve the objectives of attack accuracy and invisibility.

The lLOC attack can not only attack *one* target network to determine the deployment of monitors in *one* network at one time, but it can also attack *multiple* networks simultaneously. Intuitively, one simple way to achieve this parallel attack is to launch port-scan/attack traffic towards multiple target networks simultaneously, by scanning a different port number for each different target network. For example, if the data center publishes traffic reports of 1000 (TCP/UDP) ports, then the attacker can launch attacks towards 1000 networks simultaneously, attacking each network with a different port number. Since attack traffic on different ports are summarized separately at the data center, the attacker still can separate and thus decode its traffic

towards different targets. Hence, the attacker can localize monitors in multiple networks simultaneously and accurately. However, can the attacker further improve the attack efficiency? Assuming that the data center still only publishes reports of 1000 ports, can the attacker fingerprint 10,000 target networks simultaneously, for example, by attacking 10 different networks using *the same* port number? Using a high-rate of port-scan traffic cannot achieve this, because it is indiscernible whether a spike in the traffic report is caused by traffic logs from one network or the other 9 networks. In order to achieve this goal in the code-based attack, the selected code and corresponding encoded attack traffic towards multiple networks for the same port should not interfere with each other (i.e., each of them can be decoded *individually* and *accurately* by the attacker, although they are integrated/summarized in the traffic report from the ITM data center). The PN-code selected in the lLOC attack has this feature, giving it the unique capacity to carry out parallel attack sessions towards multiple target networks using *the same* port. The details of the PN-code selection will be discussed in the following sections. In the following, we will give the details of attack stages illustrated in Fig. IV-1.

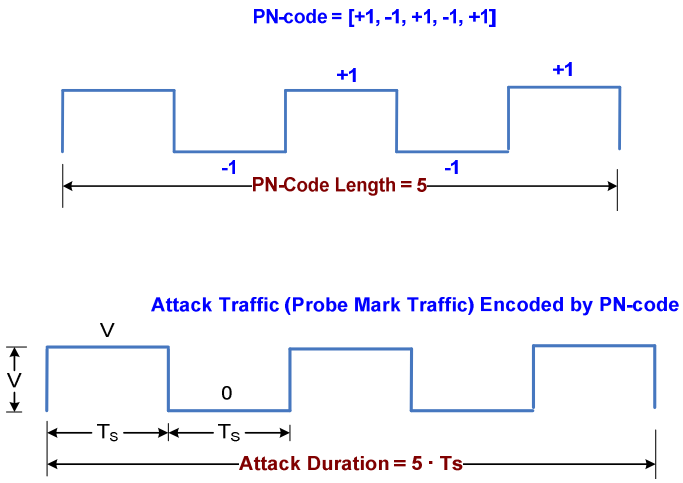


Fig. IV-2. PN-code and Encoded Attack Traffic

2.b. Attack Traffic Generation Stage

In this attack stage, the attacker: (i) selects the code, a *PN-code* in our case; (ii) encodes the attack traffic using the selected PN-code; and (iii) launches the encoded attack traffic towards the target network. For the third step, the attacker can coordinate a large number of compromised bots to launch the traffic [7]. However, this is not the focus of this chapter. In the following, we will present detailed discussion of the first and second steps, respectively.

1) Code Selection

To evade detection by others, the attack traffic should be similar to the background traffic. From a large set of real-world background traffic traces obtained from SANs ISC [25, 52], we conclude that the background traffic shows random patterns in both time and frequency domains. The attack objectives of both accuracy and invisibility, and an attacker's desire for parallel attacks require that: (i) the encoded attack traffic should blend in with background traffic, i.e., be random in both the time and frequency domains, (ii) the code embedded in the attack traffic should be easily recognizable to the attacker himself, and (iii) the code should support parallel attacks.

To meet the above requirements, we choose the PN-code to encode the attack traffic. The PN-code in the lLOC attack is a sequence of -1 or +1 with the following features [53, 54, 55]. The PN-code is random and "balanced". The -1 and +1 are randomly distributed and the occurrence frequencies of -1 and +1 are nearly equal. This feature contributes to good spectral density properties (i.e., equally spreading the energy over the whole frequency-band). It makes the attack traffic appear as noise and blend in with background traffic in both time and frequency domains.

The PN-code has a high correlation to itself and a low correlation to others (such as random noise), where the correlation is a mathematical tool for finding repeating patterns in a signal

[55]. This feature makes it feasible for the attacker to accurately recognize attack traffic (encoded by the PN-code) from the traffic report data even under the interference of background traffic.

The PN-code has a low cross-correlation value among different PN-code instances. The lower this cross-correlation, the less interference among multiple attack sessions in parallel attack. This feature makes it feasible for the attacker to conduct parallel localization attacks towards multiple target networks on the same port.

The Walsh-Hadamard code and M-sequence code [53, 54] are two popular types of PN-code. The Walsh-Hadamard code has some limitations. Since its frequency spreads into only a limited number of discrete frequency components, which is different from background traffic, it will compromise the invisibility of the attack traffic if used in the lLOC attack. In addition, the Walsh-Hadamard code also strongly depends on global synchronization [54]. To the contrary, M-sequence code does not have these shortcomings, so we adopt M-sequence codes in the lLOC attack. We use the *feedback shift register* to repeatedly generate the M-sequence PN-code due to its popularity and ease of implementation [53, 56]. In particular, a feedback shift register consists of two parts. One is an ordinary shift register consisting of a number of flip-flops (two state memory states). The other is a feedback module to form a multi-loop feedback logic.

2) Attack Traffic Encoding

During the attack traffic encoding process, each bit in the selected PN-code is mapped to a unit time period T_s, denoted as mark bit duration. The entire duration of launched traffic (referred to as traffic launch session) is T_sL, where L is the length of the PN-code. The encoding is carried out according to the following rules: each bit in the PN-code maps to a mark bit duration (T_s); when the PN-code bit is +1, port-scan traffic with a high rate, denoted as mark traffic rate μ, is generated in the corresponding mark bit duration; when the code bit is -1, no port-scan traffic is

generated in the corresponding mark bit duration. Thus, the attacker embeds the attack traffic with a special pattern, i.e., the original PN-code.

Recall that, after this encoding process, the PN-code pattern embedded in traffic is denoted as attack mark. If we use $c_i =< c_{i,1}, c_{i,2}, \ldots, c_{i,L} >$ in $\{-1,+1\}^L$ to represent the PN-code and use $\eta_i =<\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,L}>$ to represent the attack traffic, then we have $\eta_{i,j}= \mu/2 \cdot c_{i,j} + \mu/2$ ($j = 1,\ldots, L$). Fig. IV-2 shows an example of the PN-code and the corresponding attack traffic encoded with the PN-code.

2.c. Attack Traffic Decoding Stage

In this stage, the attacker takes the following two steps: (i) The attacker queries the data center for the traffic report data, which consists of both attack traffic and background traffic. (ii) From the report data, the attacker attempts to recognize the embedded attack mark. The existence of the attack mark determines the deployment of monitors in the attack targeted network. As the query of traffic report data is relatively straightforward, here we only detail the second step, i.e., attack mark recognition, as follows.

In the report data queried from the data center, the attack traffic encoded with the attack mark is mixed with background traffic. It is critical for the lLOC attack to accurately recognize the attack mark from the traffic report data. To address this problem, we develop the correlation-based scheme. This scheme is motivated by the fact that the original PN-code (used to encode attack traffic) and its corresponding attack mark (embedded in the traffic report data) are highly correlated; in fact, they are actually the same.

The attack mark in the traffic report data is the *embedded form* of the original PN-code. The attack mark is similar to its original PN-code, although the background traffic may introduce interference and distortion into the attack mark. We adopt the following correlation degree to

measure their similarity. Mathematically, the correlation degree is defined as the inner product of two vectors. For two vectors $X=<X_1, X_2, \ldots, X_L>$ and $Y=<Y_1, Y_2, \ldots, Y_L>$ of length L, the correlation degree of vector X and Y is $\Gamma(X, Y) = X \circ Y = \Sigma_1^L X_i \cdot Y_i / L$, where $\Gamma(.)$ represents the operator for the inner product of two vectors. Based on above definition, we have $\Gamma(X, X) = \Gamma(Y, Y)=1$, if X, Y in $\{-1,+1\}^L$.

We use two vectors, $\eta_i=<\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,L}>$ and $\omega_i=<\omega_{i,1}, \omega_{i,2}, \ldots, \omega_{i,L}>$ to represent attack traffic (embedded with attack mark) and background traffic, respectively. We *shift* the above two vectors by subtracting the mean value from the original data, resulting in two new vectors, $\eta'_i=<\eta'_{i,1}, \eta'_{i,2}, \ldots, \eta'_{i,L}>$ and $\omega'_i=<\omega'_{i,1}, \omega'_{i,2}, \ldots, \omega'_{i,L}>$. We still use a vector $c_i =< c_{i,1}, c_{i,2}, \ldots, c_{i,L} >$ in $\{-1,+1\}^L$ to represent the PN-code. Thus, the correlation degree between the PN-code and the (shifted) attack traffic can be obtained. Similarly, we can also obtain the correlation degree between the PN-code and the (shifted) background traffic as follows.

According to the rules of encoding attack traffic in Section 2.3.1, $\eta_i= \mu/2 \cdot c_i + \mu/2$ and $E(\eta_{i,j})= \mu/2$. Thus, $\eta_i'= \eta - E(\eta_{i,j})= \mu/2 \cdot c_i$. Hence, the correlation degree between the original PN-code and the (shifted) probe mark embedded attack traffic is $\Gamma(c_i, \eta'_i) = \mu/2 \cdot \Gamma(c_i, c_i) = \mu/2$. Furthermore, we can also derive the correlation degree between the PN-code and the (shifted) background traffic, i.e., $\Gamma(c_i, \omega'_i)$. Since the PN-code has low correlation with the (shifted) background traffic, the mean of such correlation degree can be derived by

$$E[\Gamma(c_i, w_i^{'})] = \frac{1}{L} E[\sum_{j=1}^{L} (w'_{i,j} \cdot c_{i,j})] \approx 0. \qquad \text{(IV-1)}$$

If the standard deviation of the background traffic rate is σ_x, the variance of such correlation degree is

$$Var[\Gamma(c_i, w_i^{'})] = E[\Gamma(c_i, w_i^{'}) - 0]^2]$$
$$= \frac{1}{L^2} E[\sum_{j=1}^{L} (w'_{i,j} \cdot c_{i,j})^2] = \frac{1}{L^2} E[\sum_{j=1}^{L} (w'_{i,j})^2] = \frac{\sigma_x^2}{L}. \quad \text{(IV-2)}$$

Thus, the average correlation degree between the PN-code and the (shifted) background traffic is $\Gamma(c_i,\omega_i^{'}) = \sigma_x/L^{1/2}$. Based on the above discussion, the attacker can set appropriate attack parameters (e.g., PN-code length L and mark traffic rate μ) to make correlation degree (μ/2) between the PN-code and the attack mark traffic that is much larger than the correlation degree ($\sigma_x/L^{1/2}$) between the PN-code and the background traffic. As such, the attacker can accurately distinguish the attack mark traffic from the background traffic.

In the practice of attack mark recognition, vector λ_i is used to represent the queried report data, and vector $\lambda_i^{'}$ is used to represent the *shifted* report data (by subtracting $E(\lambda_{i,j})$ from λ_i). The attacker uses the correlation degree between $\lambda_i^{'}$ and his PN-code c_i, i.e., $\Gamma(c_i, \lambda_i^{'})$, to determine the existence of PN-code in the report data. If $\Gamma(c_i, \lambda_i^{'})$ is larger than a threshold T_a, which is referred to as *mark decoding threshold*, then the attacker determines that the report contains attack traffic as well as the PN-code c_i, and determines that the target network is deployed with monitors. The accuracy of this correlation-degree-based PN-code recognition is analyzed and demonstrated in Section 2.e.

2.d. Attack Traffic Synchronization

In order to accurately and effectively recognize the attack mark (PN-code) from the report data, we need to find the segment of the report data containing the PN-code (i.e., we need to fulfill the synchronization between the port-scan traffic report data and the PN-code). For this purpose, we introduce an iterative sliding window-based scheme. The basic idea is to let the attacker obtain enough report data with small granularity. Then, a sliding window iteratively moves forward to capture a segment of the report data. For each segment, we apply the correlation-based scheme

discussed in Section 2.c to recognize whether or not the attack mark exists. The details of this synchronization is presented as follows.

The attacker first sends a sequence of queries to the data center and each query requests a portion of report data which lasts for a given unit time, known as query duration T_q. To guarantee good synchronization and capture of each bit in the PN-code, T_q should be smaller than the mark bit duration T_s. Also, the attacker needs to send enough queries and ensure that the queried report data contains the whole attack mark and attack mark traffic, which is length μT_s. With the report data, the attacker iteratively conducts a correlation test on the report data, using a sliding window. For example, in the i-th round, the attacker selects t_i as the starting time for the sliding window. In (i+1)-th round, the attacker moves the sliding window one step (T_q) forward, thus the start time of the sliding window becomes $t_i + T_q$, and so on. In the i-th round, a sequence of data (length of L) is obtained in the sliding window. The first data point in the sequence is the traffic data in time duration $[t_i, t_i + T_s]$, the second data point in the sequence is the traffic data in time duration $[t_i+T_s, t_i+2T_s]$, and so on. With these data, the attacker conducts the attack mark recognition procedure discussed in Section 3. The attacker repeats the attack mark recognition after each time he moves forward the sliding window, until the attack mark is recognized from the report data in the current sliding window, or the sliding window has gone through all the report data.

2.e. Analysis

In this section, we first present our analysis of the impacts of different attack parameters on attack accuracy. We then discuss how to determine attack parameters.

1) Attack Accuracy Analysis

In order to measure attack accuracy, we introduce the following two metrics. The first one is *attack successful rate* P_D, which is the probability that an attacker correctly recognizes the fact that a selected target network is deployed with monitors. The higher P_{AD} is, the higher the attack accuracy. The second metric is *attack false positive rate* P_{AF}, which is the probability that the attacker mistakenly declares a target network as one deployed with monitors. The lower P_{AF}, the higher the attack accuracy is. In order to ensure attack invisibility, the obvious method is to use the low traffic rate μ. Recall that T_a is the mark decoding threshold, μ is the mark traffic rate, vector λ_i represents the queried report data, and vector λ'_i represents the *shifted* report data (by subtracting $E(\lambda_{i,j})$ from λ_i). Assume that random variables $\omega'_{i,1}$, … , $\omega'_{i,L}$ (i.e., the shifted background traffic) are independently, identically distributed (i.i.d) and follow a Gaussian random distribution with standard deviation σ_x, then we have the following theorem for the attack accuracy of the lLOC attack.

Theorem IV-1. In the lLOC attack, the attack successful rate P_{AD} is

$$P_{AD} = 1 - \Pr[\Gamma(\lambda_i^{'}, c_i) \le T_a \mid (\lambda_i^{'} = \eta_i^{'} + \omega_i^{'})] = 1 - \frac{1}{\sqrt{\pi}} \int_{\frac{(\mu/2 - T_a)\sqrt{L}}{\sqrt{2}\sigma_X}}^{\infty} e^{-y^2} dy. \tag{IV-3}$$

The attack false positive rate P_{AF} is

$$P_{AF} = \Pr[\Gamma(\lambda_i^{'}, c_i) \le T_a \mid (\lambda_i^{'} = \omega_i^{'})] = 1 - \frac{1}{\sqrt{\pi}} \int_{\frac{T_a\sqrt{L}}{\sqrt{2}\sigma_X}}^{\infty} e^{-y^2} dy. \tag{IV-4}$$

Proof:

i) Derivation of attack successful rate P_{AD}.

According to the definition of P_{AD}, we have

$$P_{AD} = 1 - \Pr[\Gamma(\lambda_i^{'}, c_i) \le T_a \mid (\lambda_i^{'} = \eta_i^{'} + \omega_i^{'})]. \tag{IV-5}$$

Consider that $\Gamma(c_i, \eta_i^{'}) = \mu/2 \Gamma(c_i, c_i) = \mu/2$, the Equation (IV-5) can be rewritten by

$$P_{AD} = 1 - \Pr[\Gamma(\lambda_i^{'}, c_i) \le T_a - \frac{\mu}{2} | (\lambda_i^{'} = \omega_i^{'})]. \quad \text{(IV-6)}$$

Based on the mean and variance of correlation degree determined in Section 3, P_{AD} is represented by

$$P_{AD} = 1 - \frac{\sqrt{L}}{\sqrt{2\pi}\sigma_X} \int_{-\infty}^{T_a - \frac{\mu}{2}} e^{\frac{-x^2 L}{2\sigma_X{}^2}} dx. \quad \text{(IV-7)}$$

Let $y^2 = x^2 L / 2\sigma_x^2$, then we have

$$P_{AD} = 1 - \frac{\sqrt{L}}{\sqrt{2\pi}\sigma_X} \int_{-\infty}^{\frac{(T_a - \frac{\mu}{2})\sqrt{L}}{\sqrt{2}\sigma_x}} \frac{\sqrt{2}\sigma_X}{\sqrt{L}} e^{-y^2} dy = 1 - \frac{1}{\sqrt{\pi}} \int_{\frac{(\frac{\mu}{2} - T_a)\sqrt{L}}{\sqrt{2}\sigma_X}}^{\infty} e^{-y^2} dy. \quad \text{(IV-8)}$$

ii) Derivation of attack false positive rate P_{AF}

We know that $\Gamma(c_i, \lambda_i^{'}) = \lambda_I^{'} \circ c_i$, *where* $\lambda_i^{'} = \omega_i^{'}$ when no lLOC attack traffic exists. Assuming that $\Gamma(c_i, \lambda_i^{'})$ follows a Gaussian distribution $N(0, \sigma_x^2/L)$ (discussed in Section 3), we have

$$P_{AF} = \Pr[\Gamma(\lambda_i^{'}, c_i) \ge T_a | (\lambda_i^{'} = \omega_i^{'})]. \quad \text{(IV-9)}$$

Thus, P_{AF} can be presented by

$$P_{AF} = 1 - \frac{\sqrt{L}}{\sqrt{2\pi}\sigma_X} \int_{T_a}^{\infty} e^{\frac{-x^2 L}{2\sigma_X{}^2}} dx. \quad \text{(IV-10)}$$

Letting $y^2 = x^2 L / 2\sigma_x^2$, then we have

$$P_{AF} = \frac{\sqrt{L}}{\sqrt{2\pi}\sigma_X} \int_{\frac{T_a\sqrt{L}}{\sqrt{2}\sigma_x}}^{\infty} \frac{\sqrt{2}\sigma_X}{\sqrt{L}} e^{-y^2} dy = 1 - \frac{1}{\sqrt{\pi}} \int_{\frac{T_a\sqrt{L}}{\sqrt{2}\sigma_X}}^{\infty} e^{-y^2} dy. \quad \text{(IV-11)}$$

Remarks: We make a few observations based on the theorem presented above. First, the attack successful rate P_{AF} increases and the attack false positive rate P_{AF} decreases with increasing PN-

code length L. That is, higher attack accuracy increases when L increases. Second, with the increasing mark traffic rate μ, attack accuracy also increases.

2) Determination of Attack Parameters

(a) Determination of μ, T_a and L: The attacker can determine the values of attack parameters based on the above analysis. First, the attacker can determine the mark traffic rate μ based on the statistical knowledge for the background traffic. Given the μ, the attacker can further determine the mark decoding threshold T_a and PN-code length L. Note that the values of other attack parameters such as the standard deviation of background traffic σ_x can be determined through analyzing historical background traffic data published by the data center of the ITM system.

(b) Mark recognition threshold T_a: Given the mark traffic rate μ (determined previously) and desired attack false positive rate P_{AF}, the attacker can further determine the mark decoding threshold T_a by resolving Equation (IV-9) in Theorem IV-1.

(c) Length of PN-code L: Given the mark traffic rate μ, mark decoding threshold T_a, and desired attack successful rate P_{AD}, the attacker can further determine the length of PN-code L by resolving (IV-3) in Theorem IV-1.

(d) Determination of T_s: To determine the mark bit duration T_s, the attacker needs to estimate the possible delay from the moment the attack traffic is first reported by monitors, to the moment when such attack traffic is published by the data center. To make the lLOC attack effective, the mark bit duration needs to be at least as large as such delay. Otherwise, the traffic in different bit durations (each last T_s) may be published at the same moment from the data center, mixing and thereby rendering them inseparable.

Several possible methods can be used to obtain such delay information. Some ITM systems may publish such information on their websites. The attacker may also actively conduct experiments on ITM systems and measure such delay. For example, the attacker may deploy

monitors in his controlled (small) network and connect them to the targeted ITM system. The attacker can simply use such monitors to report logs embedded with special patterns (e.g., PN-code) and keep querying the data center until the embedded traffic patterns are recognized. After repeating the above process several times, the attacker is able to obtain the statistics profile of delay information, and then determine the mark bit duration T_s. We use this method in our implementation of the lLOC attack, which is presented in the next section.

3. Performance Evaluation of lLOC Attacks

3.a. Evaluation Methodology

In our evaluation, we use the real-world port-scan traces from SANs ISC (Internet Storm Center) including the detail logs from 01/01/2005 to 01/15/2005 [25, 52]. The traces used in our study contain over 80 million records and the overall data volume exceeds 80 GB. We use these real-world traces as the background traffic. We merge records of simulated lLOC attack traffic into these traces and replay the merged data to emulate the lLOC attack traffic. We evaluate different attack scenarios by varying attack parameters. Here, we only show the data on port 135; experiments on other ports result in similar observations.

We explore both attack accuracy and invisibility to evaluate attack performance. For attack accuracy, we use two metrics: one is the *attack successful rate* P_{AD} and the other is the *attack false positive rate* P_{AF}, which are defined in Section 5. For attack invisibility, we use two metrics: one is the *defender detection rate* P_{DD} and the other is *defender false positive rate* P_{DF}. For the countermeasure, we only use a representative and generic algorithm which has no specific requirement on detection systems. More comprehensive countermeasures will be studied in Section 4. This simple threshold-based detection algorithm is widely adopted by many systems [2, 18, 25, 31]. In this algorithm, if the traffic rate (volume in a given time duration) is

larger than a pre-determined threshold T_d (referred to as the *defender detection threshold*), the defender issues threat alerts and initiates reactions [25]. Such a detection threshold is usually obtained through statistical analysis of the background traffic. Note that the threshold T_d must be carefully chosen for anomaly detection: it must maintain both high detection rate (i.e., the probability that an ongoing attack is detected) and low false positive rate (i.e., the probability that an alarm is triggered when no attack is occurring).

We evaluate the lLOC attack in comparison with two other baseline attack schemes. The first one is the localization attack that launches a significantly high-rate of port-scan traffic to target networks as introduced in [27, 28]. We denote this attack as a *volume-based attack*. The second baseline scheme embeds the attack traffic with a unique frequency pattern. In this attack, the attack traffic rate changes periodically. Then, the attacker expects the report data from the data center to show such a unique frequency pattern if the selected target network is deployed with monitors. We denote this attack scheme as a *frequency-based attack*. For fairness, we adjust the detection thresholds in all schemes so that reasonable *attack false positive rate* P_{AF} and *defender false positive rate* P_{DF} (below 1%) are achieved. For the lLOC attack, we generate different attack traffic based on variant PN-code length L (i.e., 15, 30, 45). The default PN-code length is set to 30. To better quantify the attack traffic rate for the lLOC attack and other attack schemes, we use the normalized attack traffic rate P, which is defined as $P = \mu/\sigma_x$ for lLOC attack, where σ_x is the standard variation of background traffic rate. The default value of $T_q = 0.1T_s$. In all simulation figures, the attack traffic rate (x-axis) is based upon this normalized attack traffic rate defined above.

3.b. Evaluation Results

In this section, we will present the evaluation results.

(a) Attack Accuracy: To compare the attack accuracy of the lLOC attack with that of volume and frequency-based attack schemes, we plot the attack successful rate P_{AD} under different attack traffic rates (i.e., P in [0.01, 3]) as shown in Fig. IV-3. From this figure, we observe that both lLOC and frequency-based attacks consistently achieve a much higher attack successful rate P_{AD} than the volume-based scheme. This difference in P_{AD} is more significant when the attack traffic rate is lower, which can be explained as follows. For the lLOC scheme, the PN-code-based encoding/decoding makes the recognition of attack marks robust to interference of the background traffic. For the frequency-based scheme, the invariant frequency in the attack traffic is also robust to the interference of the background traffic. Both of them can distinguish their attack traffic accurately even when the attack traffic rate (i.e., P) is small. Nevertheless, the volume-based scheme relies on the high rate of attack traffic (i.e., large P), and thus, is very sensitive to the interference of the background traffic.

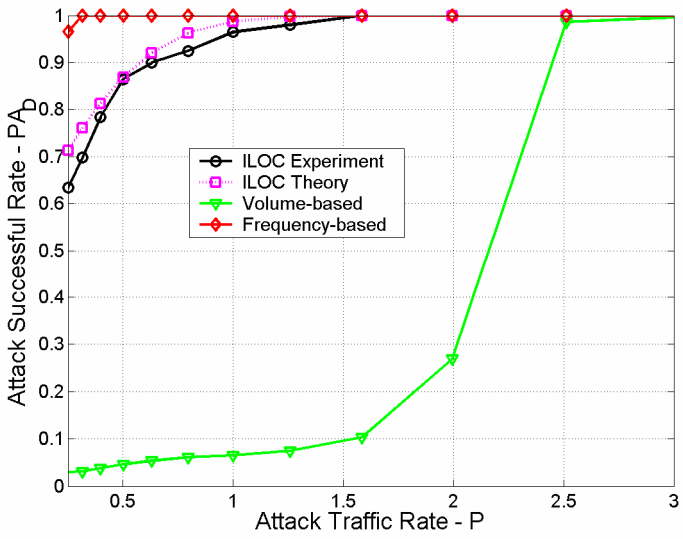


Fig. IV-3. Attack Successful Rate (Port 135)

(b) Attack Invisibility: To compare the attack invisibility performance of the lLOC attack with the other two attack schemes, we show the defender detection rate P_{DD} on port 135 in Table IV-

1. This table shows the attacker-achieved defender detection rate P_{DD}, given different localization successful rates P_{AD} (90%, 95%, and 98%). Recall that the defender sets the detection threshold to make the defender false positive rate P_{DF} below 1%. In the table, "(Time)" and "(Freq)" mean that the defender adopts the *time-domain* and *frequency-domain* analytical techniques to detect attacks. It is observed that our lLOC scheme consistently achieves much lower defender detection rate P_{DD} than other two schemes, which means the lLOC attack achieves the best attack invisibility performance. As expected, the defender can easily detect the frequency-based attack by frequency-domain analytical technique, as there is a unique frequency pattern in its attack traffic.

Table IV-1. Defender Detection Rate PD_D (Port 135)

PA_D	lLOC (Time)	lLOC (Freq)	Volume-based attack (time)	Frequency-based attack (freq)	Frequency-based attack (time)
90%	2.5%	2.2%	90%	90%	2.9%
95%	2.8%	2.4%	95%	95%	3.1%
98%	3.1%	2.8%	98%	98%	3.3%

(c) Impact of the Length of PN-code: To investigate the impact of the PN-code length on the performance of the lLOC attack, we plot the attack successful rate P_{AD} for PN-code of different lengths (15, 30, 45) in Fig. IV-4. In the legend, lLOC($L = x$) means that the PN-code length is x. Data in this figure are also collected for various attack traffic rates. This figure shows that the attack successful rate P_{AD} increases with larger PN-code length. This is because a longer PN-

code can more significantly reduce the interference impact from the background traffic on recognizing the attack mark, thereby achieving higher attack accuracy.

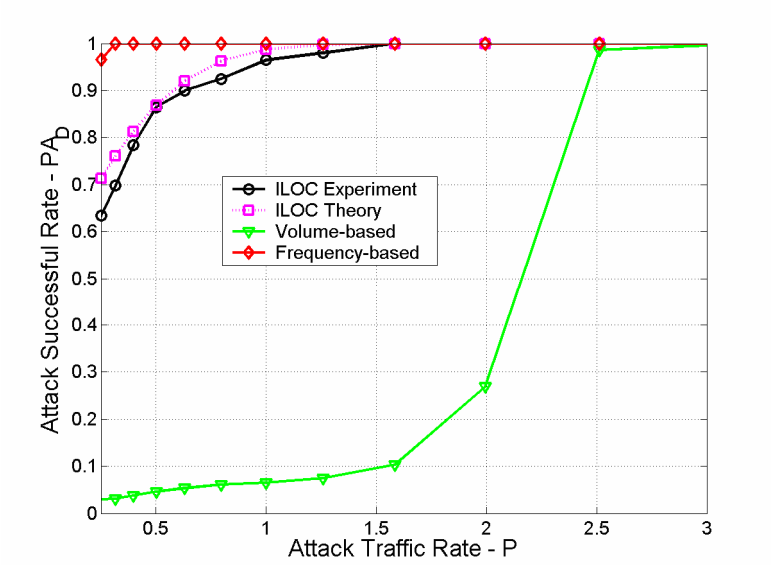


Fig. IV-4. Attack Successful Rate vs. Code Length

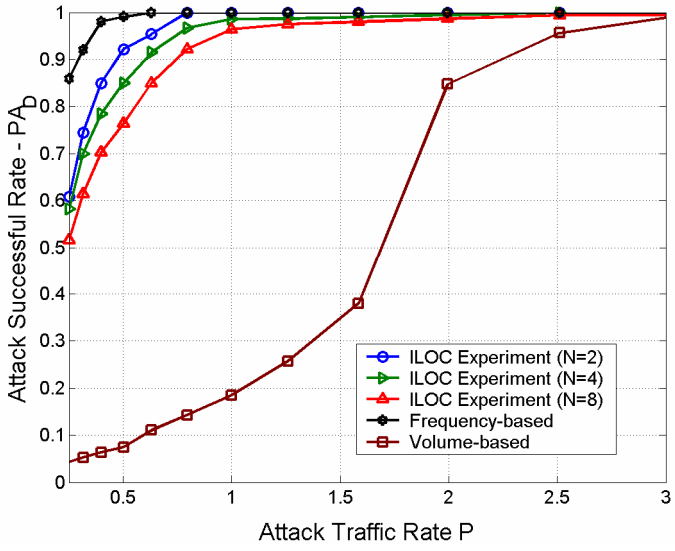


Fig. IV-5. Attack Successful Rate vs. Number of Parallel Attack Sessions

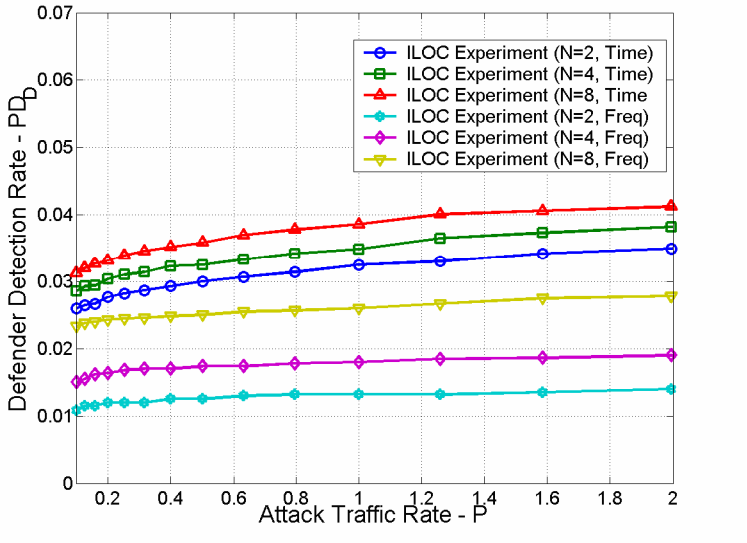


Fig. IV-6. Defender Detection Rate vs. Number of Parallel Attack Sessions

(d) Impact of the Number of Parallel Localization Attacks: To evaluate the impact of the number of parallel localization capability on attack accuracy, we show the attack successful rate P_{AD} for a different number of parallel attack sessions on the same port in Fig. IV-5. In the legend, lLOC($N = x$) means that there are x parallel attack sessions. This figure shows that in terms of attack successful rate P_{AD}, the lLOC attack scheme is not sensitive to the number of parallel attack sessions. The attack successful rate P_{AD} only slightly decreases with the increasing number of parallel attack sessions. This is because the traffic for different attack sessions are encoded by PN-codes, which are low cross-correlated to each other as described in Section 2, and thereby experience little interference. Fig. IV-6 shows the impact of the number of parallel attack sessions on attack invisibility. It can be observed that the increasing number of parallel attack sessions results in a slight increase of defender detection rate P_{DD}. Therefore, parallel localization capability can improve the attack efficiency without significantly compromising both accuracy and invisibility.

The lLOC attack achieves invisibility by using the PN-code, which contributes to a longer period during which the attack can be carried out. Nevertheless, parallel capability can significantly improve the attack efficiency. For example, let's consider the case in which a system consisting of 1200 networks is attacked. Using one port, the volume-based attack needs 1200 unit time to perform the attack task. Single lLOC attack with code length of 15 needs 1200×15 = 18000 unit time and achieves higher accuracy and invisibility. To fulfill the same localization attack task, parallel lLOC with 8 attack sessions and the same code length can achieve similarly high accuracy and invisibility performance and the total time is only 1200×15/8 = 2250 unit time, which is comparable to that of a volume-based attack.

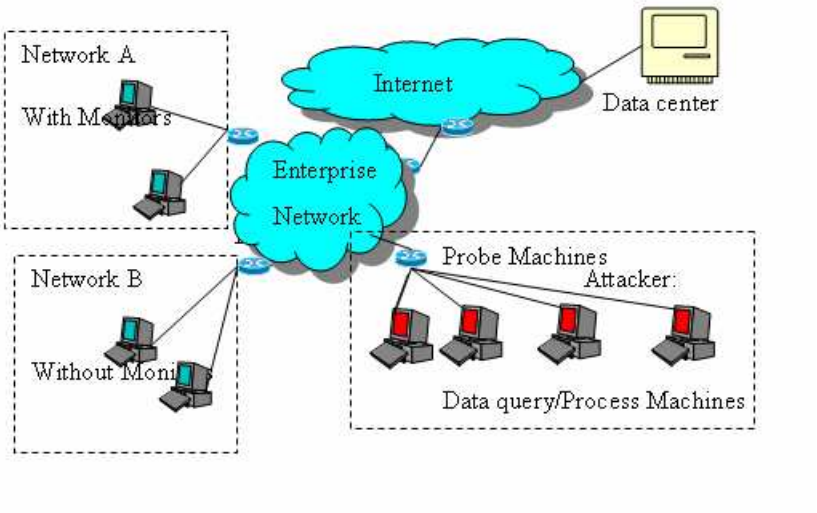


Fig. IV-7. Experiment Setup

3) Implementation and Validation

To validate the feasibility of lLOC in real-world, we introduce our implementation of the lLOC attack and report the validation results of our lLOC attack design and experiments against

a real-world ITM system. We implement an lLOC attack prototype based on the design in Section 2. This prototype works against any ITM system with the data center having a web-based user interface. Particularly, there are five independent and important components in our lLOC implementation, *Data Center Querist*, *Background Traffic Analyzer*, *PN-code Generator*, *Attack Traffic Generator* and *Attack Mark Decoder*.

In particular, *Data Center Querist* is a component that interacts with the data center of the targeted ITM system. Its main tasks consist of sending queries to the data center for port-scan traffic report and retrieving the response (i.e., the report) from the data center. The inputs to this component are the URL, or IP address, of the data center and the port number of the port-scan traffic needed to perform the query. From the traffic report data, *Background Traffic Analyzer* can obtain the statistics profile of background traffic and determine attack parameters for other components. *PN-code Generator* is a component that generates and stores the PN-code. The PN-code length is determined according to the attacker's objectives and background traffic profile as described in Section 2.e. *Attack Traffic Generator* is a component that generates attack traffic based on the PN-code and background statistics profile. In this, the PN-code encoded traffic is generated in the way discussed in Section 2.b. Inputs to this component are the IP addresses' range of target network, port number and transportation protocol (TCP or UDP). *Attack Mark Decoder* is a component that obtains the port-scan report data through *Data Center Querist*, and decides whether the attack mark exists in the way discussed in Section 2.c. The PN-code used in the decoding process is the same as the one used in encoding attack traffic and stored in the *PN-code Generator*.

These components may be integrated into one program running on one machine. The attack can also be carried out in more flexible ways if the tasks of the above components are performed

by processes on different machines. Our lLOC prototype is implemented using Microsoft MFC and Matlab on Windows XP operating system.

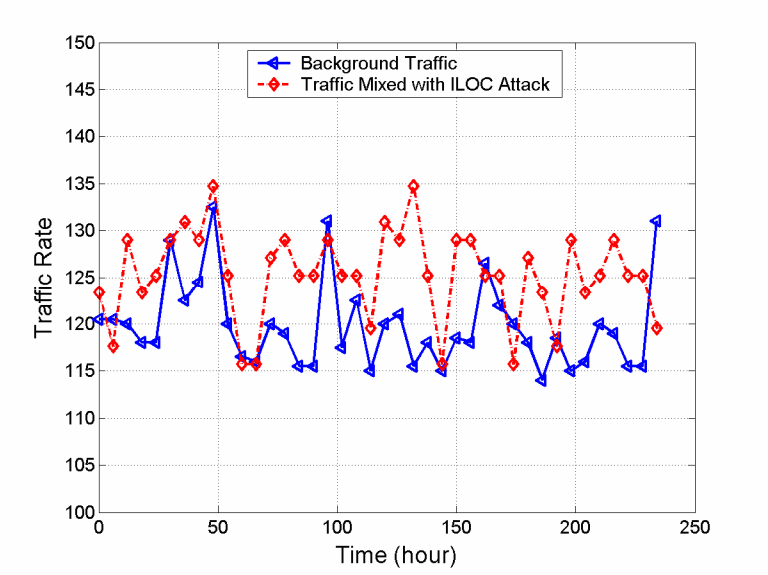


Fig. IV-8. Background Traffic vs. Traffic Mixed with lLOC Attack

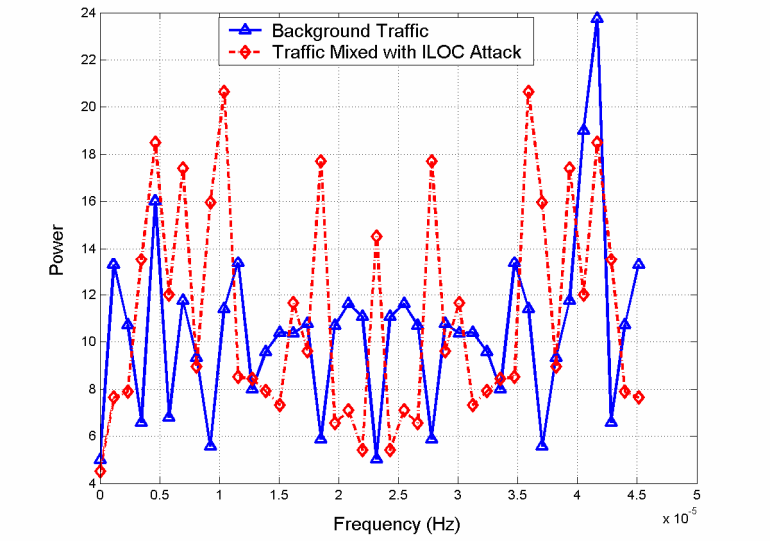


Fig. IV-9. PSD for Background Traffic vs. Traffic Mixed with lLOC Attack

In order to validate our lLOC implementation, we deployed it to identify a set of monitors that are associated with a real-world ITM system. Fig. IV-7 illustrates our experimental setup. For the purpose of this research, we requested information about locations of a set of monitors in the ITM system. We were provided with the identities of two network sets A and B. There are some monitors deployed within network set A and there is no monitor in network set B. All monitors in network set A monitor a set of IP addresses and record the port-scan logs. Then we (the attacker) execute the lLOC attack to decide whether monitors exist in network set A and set B, respectively.

In our experiment, we use a PN-code of length 15. The mark bit duration is set for 1 hour and the query duration is 20 minutes. With the queried report data, we can correctly determine that all networks in set A are deployed with monitors and networks in B are not deployed with monitors. Fig. IV-8 shows the traffic rate in time-domain. Fig. IV-9 shows the traffic rate in frequency-domain in terms of *Power Spectrum Density* (*PSD*). The *PSD* describes how the power of a time series data is distributed in frequency-domain. Mathematically, it is equal to the *Fourier* transform of the auto-correlation of time series data [57]. From these two figures, we observe that it is hard for others, without knowing the content of PN-code, to detect the lLOC attack, since the overall traffic with the lLOC attack is very similar to the traffic without the lLOC attack traffic embedded. That is, such experiments demonstrate that the lLOC attack can accurately and invisibly localize the monitors of ITM systems, in practice.

4. Countermeasure

In this section, we propose an information-theoretical based framework to explore fundamental limitations of lLOC attack strategies and develop corresponding countermeasures. We first

present the framework and then introduce the capacity derivation for measuring the system performance.

4.a. Information-Theoretical Based Framework

1) Channel Model

As shown in Fig. IV-1, an attacker launches the encoded attack traffic addressed to a target network. In order to correctly decode the embedded signal, the attacker needs to design a decoding scheme to recover his embedded signal from the background noise, which is introduced by traffic reports from other monitors not belonging to the target network. Based on the operations of localization attacks and ITM system, we can formalize the system by a channel model for digital signal transmission. In this model, the attacker (as a transmitter) generates and sends the attack signal over a noisy side channel and the attacker (as a receiver) recognizes the signal. Notice that the side channel is caused by the normal operation of ITM systems that collects data from monitors and publishes the report as shown in Fig. IV-1.

Fig. IV-10 shows the generalized channel model for the system. In particular, a source message $x = 1$ is mapped to a sequence of channel signal through the encoder. This procedure is similar to the attack stage 1 shown in Fig. IV-1.a. The output of encoder t_x is transmitted through the channel and blended with noise w, introduced by other monitors. From the channel output sequence $r_x = t_x + w$, the attacker (as a receiver) attempts to recover the transmitted message x by decoding r_x by output y. If $y \approx x$, the attacker successfully recognizes the source message x. This procedure is similar to the attack stage 2 shown in Fig. IV-1.b. By doing so, the attacker successfully determines whether the target network is deployed with monitors or not by following rules: If $y \approx 1$, the target network is deployed with monitors. Otherwise, the target network is not deployed with monitors.

Now, let's use the generalized lLOC attacks discussed in Section 2 as an example to illustrate the model, reflecting the attack as follows. First, at the transmitter, the attacker generates the source message $x = 1$ for a given network M_j. After the encoding procedure, the adversary selects an n-bit code c_t ($n \geq 1$) and generates a port-scan traffic $t_x = f_E(x, c_t, \mu') = \mu' x\, c_t$ to the network M_j, where f_E is denoted as the encoding function and μ' is mark amplitude to control the intensity of attack traffic. If the targeted network is deployed with monitors, the t_x will be transmitted through the normal operation of ITM system along with the noise w. We assume that the mean and variance of w is μ and σ, respectively. Second, at the receiver, the received signal is $r_x = t_x + w$. As the decoder procedure, it tries to decode the source message x based on the same code c_t and apply the following decision rule: If $r_x \cdot c_t = \mu' c_t \cdot c_t + w \cdot c_t \geq t_R$, then $x = 1$ and the network M_j is deployed with monitors. Otherwise, $x = 0$ and the network M_j is not deployed with monitors. Here, t_R is the decoding threshold. In order to learn how to determine the t_R, please refer to section 3.

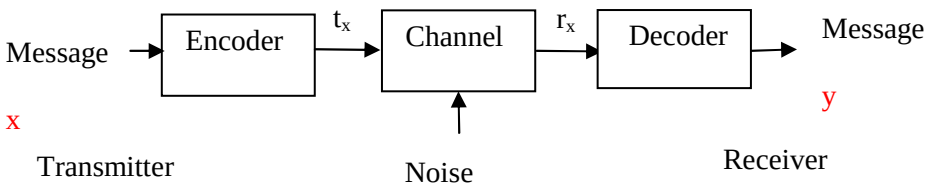


Fig. IV-10. Channel Model for lLOC Attack

In Fig. IV-10, to detect the attack, the defender will observe the output traffic r_x of channel. Recall that the defender generalizes the benign party who maintains the ITM system to identify Internet widespread attacks. Particularly, based on data stored in the data center, the defender tries to detect the anomaly in the traffic and take the mitigation.

2) Capacity

The capacity of the channel defined by Shannon provides a theoretical upper-bound for measuring the signal transmission capability over a noisy channel [58]. By definition, capacity is the amount of discrete information that can be reliably transmitted over a channel. This landmark work has been the foundation for communication system design, which aims to design various coding mechanisms to achieve the theoretical bound by various means to increase the resistance of digital signal transmission to the noise. Generally, channel coding in communication systems consists of mapping the source message into a channel input signal denoted as the encoder and the inverse mapping the channel output signal into a source message denoted as a decoder in such a way that the overall effect of channel noise on the system is minimized.

In the model described in Fig. IV.10, we denote the attack signal $t_x =< t_{x1}, t_{x2}, \ldots, t_{xn} >$ as the transmitted signal over the channel. To measure the amplitude of the transmitted signal, we define its transmission power as

$$s = \frac{1}{n}\sum_{i=1}^{n} t_{xi}^2 . \qquad \text{(IV-12)}$$

Without loss of generality, we denote the noise $w =<w_1, w_2, \ldots, w_n>$ ($n \geq 1$) with zero-mean and variance of σ. Assuming that both the signal and noise are a Gaussian white noise (WGN) process, the capacity of such a Gaussian channel is derived by,

$$C = \frac{1}{2}\log(1 + \frac{s}{\sigma}). \qquad \text{(IV.13)}$$

Since the capacity C measures the degree of successful signal transmission over the channel, the higher value of capacity denotes the better localization attack effects. From (IV-11), we know that, given the noise variance σ, a larger transmission power s will achieve a higher capacity of attack signal transmission.

As we mentioned, the capacity C in [0, 1] measures the theoretical bound for reliable signal transmission. Given any transmission error rate $\varepsilon > 0$, for any large n-bits attack signal and a minimal length length of l ($\geq n$) for $C \geq n/l$, there exists a encoding/decoding scheme, such that maximal probability of error is less than ε; that is, it is always possible to transmit the signal with arbitrarily small error, if $C \geq n/l$.

For the localization attack, the time for transmitting 1-bit attack signal is denoted as chip duration t_c. For n-bit attack signal for identifying a monitor, the minimal code length for reliable channel transmission is n/C and the minimal time for n-bit transmitted signal is nt_c/C.

Based on the information-theoretical based framework presented above, we now introduce some strategies for the attacker and defender.

(a) Attacker: Recall that for the threat model described in Section 2, the attacker intends to accurately and secretly identify monitors by launching port-scan attack traffic embedded with an attack signal. Based on the model described in Section 4.a.2, we know that an attacker should achieve a high capacity C for the accuracy of attack and also sustain a low transmission power s for the secrecy of attack. However, from (IV-13), we know that lower transmission power s will actually cause a smaller capacity. In order to address this issue, we consider that the attacker takes strategies to spread the transmission power of attack signal. The attack strategy used in Section 3 is actually one that spreads signal power into the temporal domain. In particular, regarding the temporal domain power spreading, the attack signal can be formed as a time-series traffic. As such, the signal power in each time-duration is comparatively low for preserving attack secrecy, while summarization of signal power in all time durations can be highly preserved for attack accuracy.

In summary, since the scheme proposed in [27, 28] uses an 1-bit attack signal addressed to a single monitor, we refer to this scheme as non-time-series attack, which does not spread the

transmission power of a signal in either the temporal or spacial domain. Since the code-based scheme proposed in the previous section generates a multiple-bit attack signal addressed to a single monitor, we refer to this scheme as a general attack strategy, namely the time-series attack, which spreads the transmission power of a signal into the temporal domain.

(b) Defender: To defend against localization attacks, the defender should develop countermeasures to detect attacks based on limitations of attack schemes. Based on the information-theoretical framework, the defender should develop schemes to effectively decrease the capacity. Based on (IV-13), there are two ways to decrease the capacity. One is to increase the power of noise σ. The other is to decrease the transmission power s of attack signal. However, adding noise will jeopardize the accuracy of data reported by the ITM system and degrade the usability of ITM systems. In this chapter, we will focus on developing the countermeasures that detect traffic anomaly based on the limitations of attack schemes and are able to significantly decrease the effectiveness of attacks.

To address the two attack strategies mentioned earlier (e.g., the non-time-series attack and time-series attack), we consider the following two countermeasure schemes for the defender: (1) *Centralized defense*. In this scheme, the defender will carry out anomaly detection on the centralized data center based upon the summarized traffic from all monitors in the ITM system. If the overall traffic rate (e.g., volume in a given time duration) is larger than a pre-determined threshold, the defender will issue alarms. This scheme is commonly used by existing ITM systems to defend against worm propagation and DoS attacks [25]. We will show that this countermeasure scheme is effective against the non-time-series attack in Section 5. (2) *Distributed defense*. In this scheme, each monitor will autonomously carry out defense distributedly. Each monitor will be responsible for detecting the anomaly based upon its local statistical traffic profile. If the traffic rate (e.g., volume in a given time duration) on a monitor is

larger than a pre-determined threshold for that monitor, the monitor will issue alarms. We will show that this countermeasure scheme is effective against the time-series attack in Section 5.

In the following sections, we will use our information-theoretical framework to investigate the performance of systems with the different attack and countermeasure schemes discussed above.

4.b. Defense Against lLOC Attack

In this section, we first show the *centralized defense* becomes ineffective against lLOC attack. We then introduce a new countermeasure scheme, called *advance defense*, and show that it is effective against the lLOC attack.

1) Effectiveness of Centralized Defense

We now derive the transmission power constraint of attack signal limited by centralized defense. Recall that we consider the attacker that adopts time-series attack that uses *n*-bit attack signal addressed to a single monitor M_j. For the centralized defense, the defender observes the aggregated traffic rate and compares it with a pre-known hypothesis on the distribution of background noise traffic. For the transmission power of an attack signal for the system with centralized defense, we present the following theorem.

Theorem IV-2. When the defender uses mean aggregated traffic volume of a time-series data for attack detection, in order to maintain a detection rate lower than β, the signal power s of the attacker must satisfy

$$s \le \sigma^2 \Omega_2(\beta, \alpha, n), \tag{IV.14}$$

where

$$\Omega_2(\beta, \alpha, n) = \frac{\pi e^{(1-\beta)^2} \beta^2 (1-\delta-p_0)^2}{(1-\delta)^2 (1-p_0)^2 \sqrt{n}} \tag{IV.15}$$

Proof: Suppose that the attack signal generated by an attacker at time i is μ_i'. As such, the distribution of traffic rate under attack (i.e., the combined rate of attack signal and background noise traffic) at round i is normal distribution with mean $\mu + \mu_i$'and variance σ.

Suppose that the observed traffic rate for time i is $f_M(i)$. As we can see, the observed mean traffic rate for time period [1, n] is

$$f_M(n) = \frac{1}{n}\sum_{i=1}^{k} f(i). \tag{IV-16}$$

Suppose that $f_M(n)$ is the upper bound on p- and p'-confidence interval of the background noise traffic distribution and the under-attack traffic volume distribution, respectively. According to Bayesian theorem, if the defender issues an alarm based on $f_M(n)$, the probability of a false alarm is

$$\Pr(\textit{no attack}) = \frac{(1-p_0)(1-p/2)}{p_0(1-p'/2)+(1-p_0)(1-p/2)}. \tag{IV-17}$$

Note that the attacker needs to limit the detection rate under β. In order to do so, the attacker must ensure that no alarm will be issued when $f_M(n)$ is less than or equal to the β-quantile of the under-attack traffic volume distribution. That is,

$$f_M(n) \le (\mu + \mu_n^{'*}) + \frac{\sqrt{\sigma}\Phi^{-1}(1-\beta)}{\sqrt{n}}. \tag{IV-18}$$

Where $\mu_n^{'*} = (\mu_1^{'} + \ldots + \mu_n^{'})/n$ is the mean of attacker's signal from time 1 to n.

In order to prevent the defender from issuing an alarm, the attacker must ensure that for all $f_M(n)$ that satisfies (IV-18), there is

$$\Pr\{\textit{no attack} \mid f_M(n)\} > \delta. \tag{IV-19}$$

Note that

$$\Pr\{\textit{no attack} \mid f_M(n)\} = \frac{(1-p_0)(1-p/2)}{(1-p_0)(1-p/2)+p_0\beta}, \tag{IV-20}$$

where

$$1-\frac{p}{2}=1-\Phi(\frac{\sqrt{n}\mu_n^{'*}}{\sqrt{\sigma}}+\Phi^{-1}(1-\beta)\le\beta-\frac{\sqrt{n}\mu_n^{'*}}{\sqrt{\pi}\sigma}e^{-(1-\beta)^2/2}. \quad \text{(IV-21)}$$

Thus, in order to have Pr{no attack| $f_M(n)$}>δ for all $f_M(n)$ that satisfy (IV-18), there must be

$$\mu_n^{'*}\le e^{(1-\beta)^2/2}\sqrt{\frac{\pi}{n}}\sqrt{\sigma}\,\frac{\beta(1-\delta-p_0)}{(1-\delta)(1-p_0)}. \quad \text{(IV-22)}$$

Recall that s is the power of attack signal. Due to (IV-22), with some mathematical manipulation, we can derive a power constraint as follows:

$$s\le\frac{\pi\sigma e^{(1-\beta)^2}\beta^2(1-\delta-p_0)^2}{(1-\delta)^2(1-p_0)^2\sqrt{n}}. \quad \text{(IV-23)}$$

2) Derivation of Capacity: Given the upper bound of transmission power in (IV-14), the capacity of the system becomes

$$C=\frac{1}{2}\log(1+\Omega_2(\delta,\beta,n)). \quad \text{(IV-24)}$$

Based on this, we derive the minimal code length for basic time-series attack as follows:

$$l\ge\frac{n}{C}=\frac{2n}{\log(1+\Omega_2(\delta,\beta,n))}. \quad \text{(IV-25)}$$

We now illustrate the results with practical examples. In particular, we set the parameters as follows: for the Gaussian distribution, when s = 0.44σ, the localization accuracy rate becomes 57.97% and the capacity is C = 0.06. Thus, the adversary is able to launch at least n=15 length of attack signal for both a secret and accurate attack. As a result, we know that the centralized defense scheme by itself is no longer effective against the basic time-series attack.

3) Case Study: PN-Code-Based lLOC Attack

The capacity we derive above is the theoretical bound without the detailed forms of coding and decoding scheme. Now, we conduct a case study on the code scheme investigated in Section

3. In this scheme, it adopts the simple correlation-based coding and decoding scheme. In the following, we will start with a binary channel model for this attack scheme and then present the error rate of signal transmission followed by a derivation of suboptimal capacity and observations.

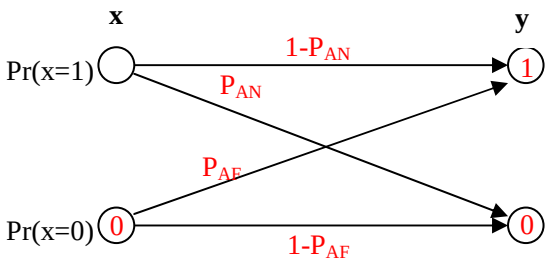


Fig. IV-11. The Binary Channel Model for PN-code Based Scheme

(a) Binary Channel Model. The binary channel model for PN-code-based scheme is shown in Fig. IV-11. Here, we represent the input of channel as a binary random variable x, where $x = 1$ represents that the targeted network is deployed with monitors, and $x = 0$ represents that the targeted network is not deployed with monitors. $\Pr(x = 1)$ and $\Pr(x = 0)$ are the prior probabilities of a network deployed with monitors or without monitors, respectively. The outputs of channel as the localization results are modeled as a random variable y, where $y = 1$ indicates that the targeted network has monitor, and $y = 0$ indicates that the targeted network has no monitor. An event has a probability $\Pr(y = 0|x = 1)$ is considered as false-negative rate denoted as ($P_{AN} = 1 - P_{AD}$) and the probability $\Pr(y = 1|x = 0)$ is considered as false-positive rate denoted as P_{AF}. Remark that P_{AD} and P_{AF} can be derived based on (IV-3) and (IV-4), respectively.

(b) Derivation of Capacity. Given the derived P_{AN} and P_{AF}, we can obtain the capacity for the code-based attack scheme. According to the definition of $I(x; y)$, we can derive the mutual information $I(x; y)$ of x and y by

$$\begin{aligned}&(1-a)(1-P_{AF})\log\frac{1-P_{AF}}{\Pr(y=0)}+aP_n\log\frac{P_{AN}}{\Pr(y=0)}+\\&(1-a)P_{AF}\log\frac{P_{AF}}{\Pr(y=1)}+a(1-P_{AN})\log\frac{(1-P_{AN})}{\Pr(y=1)},\end{aligned} \tag{IV-26}$$

where $\Pr(x = 1) = a$ and $\Pr(x = 0) = 1 - a$.

With the $I(x; y)$, the suboptimal capacity can be derived by $C = I(x; y) - H(x)$, where $H(x)$ can be derived by

$$H(x) = -\sum_{x\in(0,1)} \Pr(x)\log(\Pr(x)). \tag{IV.27}$$

4) Distributed Defense

We now consider the distributed defense. We will first derive the transmission power of the attack signal under this defense, and then derive the capacity of the system, followed by some observations.

(a) *Transmission Power of Attack Signal*: In the distributed defense, the defender carries out anomaly detection based on traffic of an individual monitor. If the traffic rate on a monitor is larger than the predetermined threshold (determined by statistical analysis of traffic from the monitor), the defender will raise threat alarms. Considering the attacker adopts the time-series attack, the transmission power of attack signal can be derived based on following theorem:

Theorem IV-3: When the defender uses the mean traffic rate on an individual monitor to carry out anomaly detection, in order to maintain a detection rate lower than β, the transmission power s of attack signal must satisfy

$$s \le \frac{\sigma}{m}\Omega_2(\delta,\beta,n), \tag{IV.28}$$

where m is the total number of monitors in the ITM system, $\Omega_2(\delta, \beta, n)$ is same as that defined in Theorem IV-2.

Proof: Recall that there are m monitors in the system and the aggregated background noise traffic is σ. Since the traffic from different monitors are independent, the traffic for individual monitor can be approximately represented by σ/m. Recalling that the defender based on the distributed defense will monitor traffic anomaly on the traffic from the individual monitor, the transmission power in (IV-28) can be derived by similar procedures in the proof of Theorem IV-2.

(b) *Capacity Analysis:* Given the transmission power of the attack signal derived in (IV-28) of Theorem IV-3, we now derive the capacity of the system where the defender uses the distributed defense and the attacker uses the time-series attack. The capacity of such system becomes

$$C = \frac{1}{2}\log(1+\frac{s}{\sigma^2}) \le \frac{1}{2}\log(1+\frac{\Omega_2(\delta,\beta,n)}{m}). \tag{IV.29}$$

Given the capacity, the minimal code length becomes

$$l = \frac{n}{C} = \frac{2n}{\log(1+\frac{\Omega_2(\delta,\beta,n)}{m})}. \tag{IV.30}$$

We now illustrate the results with practical examples. In particular, we set the system parameters as follows: for the Gaussian distribution, when m=1000 and δ=0.02, β=0.02, n=40, we can achieve capacity C=0.02. Thus, the adversary has to use a minimal 40/C=2000 length of signal to achieve accurate monitor localization while avoiding detection. However, such a long code length makes the attack scheme no longer feasible in practice. As we can see, when the defender adopts the distributed defense, the attack can no longer be effective.

4.c. Discussion

We have developed a unified information-theoretical based framework to model and analyze the localization attacks and countermeasures. There are a number of possibilities for extending this work. The detailed discussion follows:

1) Proactive Countermeasures: The countermeasure proposed in this study mainly focuses on detection. Nevertheless, other proactive countermeasures can be used. For example, limiting the information access rate on ITM systems is one way to counter attack. Recall that in the localization attack, the attacker has to launch a significant amount of queries to the data center of ITM systems in order to accurately recognize the marked attack traffic. The data center may throttle the query request rate via enforcing human/system interaction for the query, thereby eliminating the automatic query in the localization attack. Since this countermeasure increases the quantization error of the attack signal, it decreases the channel capacity of the localization attack. Perturbing the information is another way to counter the attack. Specifically, we may perturb the published report data by adding some random noise and even randomizing the data publishing delay. Since this approach increases the power of noise, the capacity of localization attack can also be decreased.

2) Spectrum-Domain Attack Schemes: Our study mainly focuses on the traffic analysis approaches in the time domain. For example, in the time-series attack, attack traffic encoding and decoding are based on the time domain; for the countermeasures, traffic anomaly analysis is also based on traffic on the time-domain. Nevertheless, this has not been true in practice. The attacker may manipulate its attack traffic in the frequency-domain. In one case, the attacker may modulate the attack traffic with a specific feature frequency. Thus, the attacker expects the report data from the data center to show high power density in the specific frequency if the targeted network is deployed with monitors. In another case, the attacker may use frequency-hop spread-

spectrum (FHSS) technique via embedding the DSSS code in the power-spectrum density of scan traffic. For both cases, our analytical methodology is still valid and can be applied via conducting analysis on the power spectrum density (PSD) of traffic. We will conduct more in-depth studies in our on-going and future work.

3) Apply to Other Systems: We focus on analyzing the localization attacks and countermeasures for a specific application. Nevertheless, our developed methodology is general and can be extended to other applications such as DSSS-based flow marking for invisible traceback, and timing delay watermarking against anonymous communication systems [59]. Since these applications correspond with different problem domains, we need to investigate the system specific information impact on the capacity, such as how accurately a flow can be marked via flow interference, how much noise for flow marking can be introduced by mix network mechanisms (i.e., flow split, merge, batching etc). We leave the detail study for our on-going and future work.

5. Performance Evaluation of Countermeasures

In this section, we present the numerical and simulation results of systems with localization attacks and countermeasures investigated in early sections. In particular, we obtain the numerical data of the capacity based on two cases: (i) the theoretical bound without considering any specific coding/decoding schemes, and (ii) one practical implementation of the correlation-based decoding scheme presented in Section 3. For the theoretical bound, we use minimal code length l to measure the performance of the system with localization attacks and countermeasures. The minimal code length is defined as the minimal length of code that the attacker has to use for the reliable transmission of attack signal. For the practical implementation, we use the code-based

attack as a specific implementation, which uses the simple correlation-based scheme discussed in Section 3.

For the practical implementation of the correlation-based upon specific coding/decoding schemes, we simulate the countermeasure performance. For the background traffic, we use the real-world port-scan traces from SANs ISC (Internet Storm Center) including the detail logs from 01/01/2005 to 01/15/2005 [25, 52]. We merge records of simulated lLOC attack traffic into these traces and replay the merged data to emulate the lLOC attack traffic. Based on the traffic profile, we determine the background traffic statistic profile and threshold values for the defender. We evaluate different scenarios by varying the attacker and defender parameters. Here, we only show the data on port 135; experiments on other ports result in similar observations.

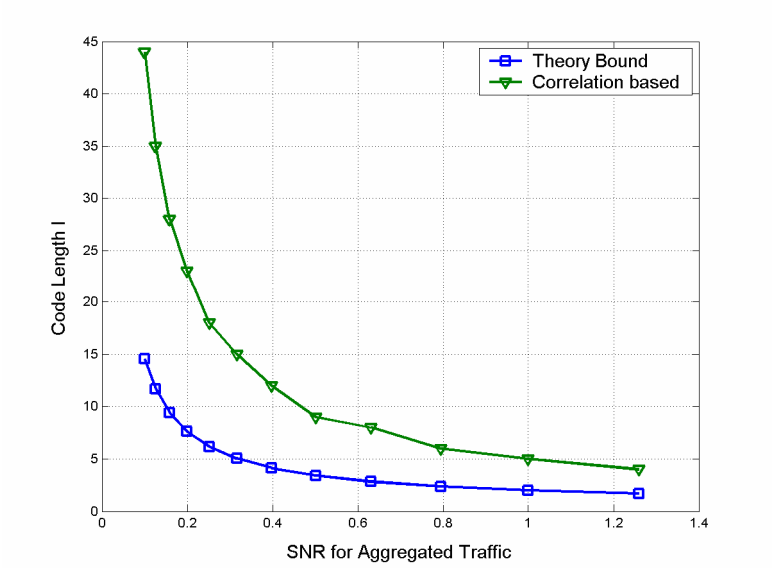


Fig. IV-12. Performance of Centralized Defense vs. lLOC Attack

To obtain the minimal code length for C-Probe attacks, the basic idea is illustrated as follows: given a high detection rate (> 99%) and low false positive rate (< 1%), we run the

simulation and find the minimal code length for a given SNR in ([0.1, 1.2]). We evaluate the performance of both the centralized defense and distributed defense against the code-based time-series attack. For the centralized defense, the SNR is the ratio of probing traffic rate over overall aggregated traffic rate on the data center. For the distributed defense, the SNR is the ratio of probing traffic rate over the traffic rate on a single monitor. The default number of monitors is 1000 and all other parameter such as δ and β are same as ones in Section 4.b and 4.c.

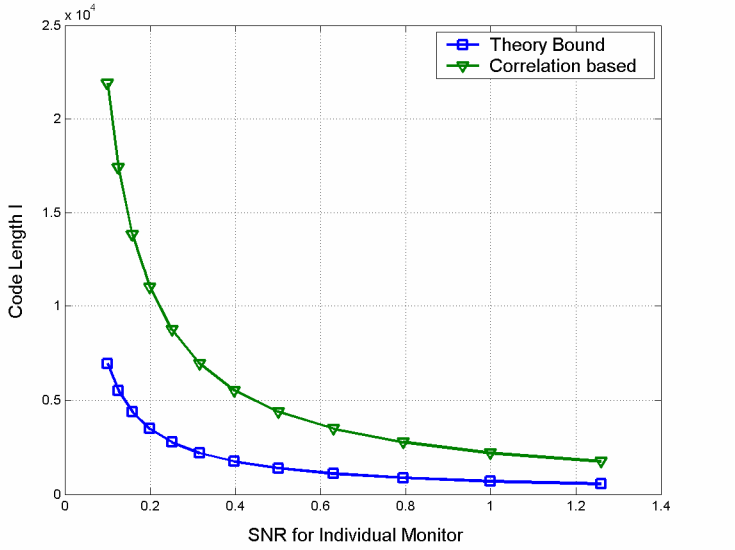


Fig. IV-13. Performance of Distributed Defense vs. lLOC Attack

Specifically, we generate different attack traffic modulated by different lengths of codes under different amplitudes of attack signal, e.g., signal-to-noise ratio (SNR), which can be observed by the defender. For example, for the centralized defense, the SNR is defined as the ratio of the transmission power of attack signal power over the variance of aggregated background noise traffic collected by ITM system. For the distributed defense, the SNR is defined as the ratio of the transmission power of the attack signal on the individual monitor and

the variance of the background traffic on the individual monitor. We obtain the localization false negative rate and false positive rate and obtain the capacity of system based on the method described in Section 3. For the code-based scheme, given a high capacity value as threshold (i.e., ≥ 0.99), we repeatedly execute the above procedures until we identify a code length that meets the requirement of large channel capacity, i.e., close to 1.

Fig. IV-12 shows the results of minimal code length vs. the SNR for the system where the attacker uses the time-series attack and the defender uses the the centralized defense. We have a few observations. First, given the reasonably small SNR (e.g, 0.2) to make the attack high invisible to the defender, the attacker is still able to use a much short length of code (e.g., $l = 15$ for correlation-based coding scheme) to accurately identify the monitors. It validates our findings that centralized defense is not effective against the time-series attack. Second, as expected, there are some performance gaps between the correlation-based coding scheme and theoretical bound. For example, when the SNR = 0.2, the correlation-based coding scheme needs to use at least length of 15 to accurately identify the monitors, while the theoretical bound indicates that code length of 9 will be enough. We believe that by incorporating other channel coding schemes such as Turbo code, we can make the performance gap smaller (close to the theoretical bound). We leave this investigation to our future work.

Fig. IV-13 illustrates the results of the minimal code length vs. SNR for the system where the attacker uses the the time-series attack and the defender uses the distributed defense. We have a few observations. First, given the reasonably small SNR (e.g, 0.2) to make the attack high invisible to the defender, the attacker must use a much longer length of code (e.g., $l = 11000$ for the correlation-based coding scheme and $l = 4500$ for the theoretical bound) to accurately identify the monitors. This validates our finding that the distributed defense is effective against

the time-series attack. Similarly, there are some performance gaps between the simple correlation-based coding scheme and theoretical bound due to the same reason illustrated earlier.

6. Summary

In this chapter, we studied the countermeasure-based on probing traffic. In particular, we investigated a new class of attacks, i.e., the low-rate lLOCalization (lLOC) attack to stealthily identify the monitors of ITM system. Its effectiveness was demonstrated via theoretical analysis, simulations and experiments with an implemented prototype. To defend against lLOC attack, we introduced an information-theoretical framework. Based on it, we derived the limitation of attack strategies and proposed the countermeasure that monitors the traffic-rate change of an individual monitor. We showed that the power constraints enforced by the countermeasure can significantly reduce the channel capacity of the system to a fairly low level that practically eliminates existing localization attacks in ITM systems. Our evaluation results effectively validated our findings. Our study is critical for securing and improving ITM systems.

CHAPTER V

COUNTERMEASURE BASED ON WORM PROGRAM EXECUTION

In the following three chapters, we will develop non-traffic based countermeasures. In this chapter, we focus on developing countermeasures based on dynamic signatures of worm program execution.

1. Overview

Many non-traffic based countermeasures have focused on static properties of worm executables [38, 39]. Specifically, in these countermeasures, the static properties such as the list of Dynamic Link Libraries (DLL) to be called, functions and specific ASCII strings extracted from the executable headers, hexadecimal sequences extracted from the executable bodies, and other static properties are used to distinguish malicious and benign executables. However, using these static properties without execution of the program might not accurately distinguish them.

It has been shown that many existing detection systems based on static properties cannot effectively detect new unseen worms which either have brand new signatures or have deliberately changed signatures during propagation [60, 61]. For example, MetaPHOR [62] and Zmist [63]) worms intensively metamorphose to hide themselves from detection. Recent studies also show that existing commercial anti-worm detection systems fail to detect brand new worms and can also be easily circumvented by worms that use simple mutation techniques [64, 65].

There are two reasons that explain why the static properties are not effective. First, two different executables (e.g. one worm and one benign) can have same static properties, e.g., they can call the same set of DLLs and even call the same set of functions. Second, these static properties can be changed by the worm writers through different ways, such as inserting dummy

functions that will not truly call during the execution in the worm executable, inserting benign looking strings, and by using code mutation tools [61, 62, 67, 68].

Hence, the static properties, or how they look, are not the keys to distinguish worm and benign executables. Instead, we believe the keys are what they do, i.e., their run-time behaviors or dynamic properties. Therefore, in this chapter we adopt dynamic program analysis to profile the run-time behavior of executables for efficiently and accurately detecting new unseen worm executables. To this end, there are three challenges to be addressed. First, we have to execute a large number of malicious worms, which might cause damage to our experiment host and network systems. Second, given the large number of executables, manually executing and analyzing them are not feasible in practice. Hence, we need to find an efficient way to automatically capture the run-time behavior from their execution. Third, from the execution of a large set of various worm and benign executables, we need to find some constant and fundamental behavior differences between the worms and the benign executables, in order to accurately determine whether an unseen executable is a worm or benign one.

To address these issues, we propose an effective worm detection approach based on mining system call traces of a large amount of real-world worms and benign executables. Our goal is to use a large volume of existing worms to capture their common dynamic signatures and then use them to detect new unseen worms. In the following, we first introduce the background and basic workflow of our approach. We then present the design detail of our approach including the dataset collection, detection feature extraction and classification, followed by the experiment results and conclusion. Notice that the work in this Chapter is based on the joined work between Texas A&M University and the Ohio State University. My work focused on the SVM data mining algorithm design, framework, and literature survey.

2. Background

In this section, we give an overview of the program analysis, data mining techniques and new unseen worms.

2.a. Program Analysis

While static program analysis requires source code of the executable, dynamic program analysis does not, but it must be performed by executing the programs [68, 69]. Most dynamic program analysis methods, such as debugging, simulation, binary instrumentation, execution tracing, stack status tracking, etc. are primarily used for software engineering and compiler optimization purposes. Recently, there has been increased attention of detecting vulnerabilities and security holes via using dynamic program analysis. However, existing dynamic analysis approaches are only suitable for analysis of individual executables with expertise such as debugging, or for specific attacks [70, 71]. However, in our case, we need an appropriate dynamic program analysis method to investigate the run-time signatures of worm and benign executables for the purpose of worm detection. The method we adopt here is to trace system calls during program execution, which is one type of light-weighted execution tracing. In particular, we trace the operating system calls invoked by the executables during their execution. This method can be used to automatically record interesting information during the execution to further investigate dynamic behavior of executables in worm detection.

2.b. Data Mining

Data mining refers to the process of extracting “knowledge,” or meaningful and useful information from large volumes of data [72, 73]. It achieves this by analyzing data from different perspectives to find inherent hidden patterns, models, relationships or any other information that

can be applied to new dataset. It includes algorithms for classification, clustering, association rule mining, pattern recognition, regression, and prediction, among others.

Data mining algorithms and tools are widely adopted in a range of application fields. In security research, many data mining technologies are adopted to conduct intrusion detection. In our work, we use the classification algorithm to obtain the difference between worm and benign program executions in order to provide accurate worm detection against both seen and un-seen worms.

There have been numerous research efforts on how to apply data mining techniques for security research [74, 75, 76, 77, 79]. For example, Lee *et at*. in [74] formulated the machine learning scheme on system call sequences of normal and anomaly execution on the Unix sendmail program. Lee *et al*. in [75] described a data mining framework for adaptively building intrusion detection models. The main tenet of their work is to utilize auditing programs (e.g., network logs of telnet sessions, shell command log) to extract an extensive set of features that describe each network connection or host session, and apply data mining techniques to learn rules that capture the behavior of intrusions and normal activities. Martin *et al*. in [76] proposed an approach via learning statistical pattern of outgoing emails from local hosts. Kolter *et al*. in [38] applied data mining techniques to extract byte sequences directly from program executables, converted these sequences into n-grams, and constructed the classifier. Julisch *et al*. in [78] proposed an approach to learn historical alarms generated by intrusion detection systems.

2.c. Unseen Worms

Although numerous efforts have been made to detect worms, the new unseen worms, including evolved forms of existing worms, can have new signatures to circumvent these existing worm detections. As we mentioned earlier, many worm detection systems use signatures of *seen*

worms to determine whether an encountered executable is worm or not. Obviously, these systems fail to detect brand new worms with new signatures and polymorphic worms that deliberately change their binary presentation or signature during propagation.

Now we will offer further discussion on polymorphic techniques [63, 80, 81]. Worms have been showing the trend to utilize these techniques for long time [61]. In particular, the technologies for mutate worm code have been publicly available even as open source toolkits or libraries [82, 83, 84]. Attackers can easily use them to make their worms polymorphic and hard to be detect by the worm detection system based on known signature. In addtion, utilizating automatic encryption and decryption further makes the polymorphism of worms more feasible and efficient. The worm detection proposed in this chapter aims to address the threat by using the dynamic properties of executable instead of static signature to capture worm executables. Since we do not use the binary presentation as the feature to distinguish worms from benign executables, the mutation techniques used by the polymorphic worms have no impact on our countermeasure scheme. As shown in the later portion of this chapter, our countermeasure based on dynamic program analysis is effective to unseen worms, including brand new worms and mutated polymorphic worms.

3. Detection via Mining Dynamic Signatures of Program Executions

3.a. Framework

1) Overview

Recall that the focus of this chapter is to use a large number of real-world worm executables and subsequently develop a countermeasure to detect new unseen worms. Now, we introduce the framework of our system for conducting dynamic program analysis, which intends to detect worm executables based on mining system call traces of a large amount of real-world worm and

benign executables. In general, this mining process is referred to as the *off-line classifier learning process*. Its purpose is to *learn* (or *train*) a generic classifier which can be used to distinguish worm executables from benign ones based on system call traces. Then, we use the learned *classifier* with appropriate classification algorithms to determine whether unknown executables belong to the worm class or the benign class with high accuracy. This process is referred to as the *on-line worm detection process*. The basic workflow is illustrated in Fig. V-1 and Fig. V-2, and explained in the following.

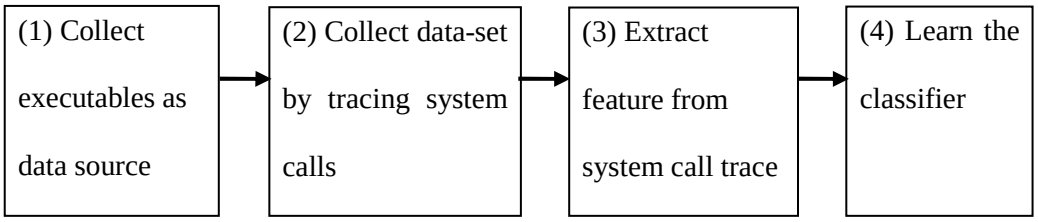


Fig. V-1. Workflow of the Off-line Classifier Learning

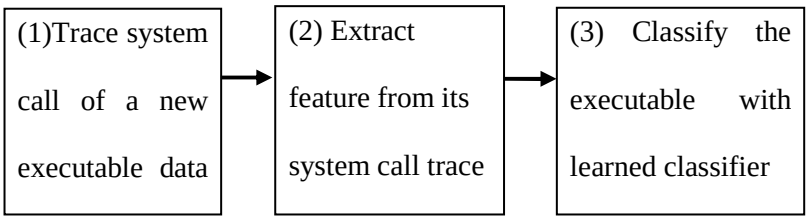


Fig. V-2. Workflow of the On-line Worm Detection

2) Off-line Classifier Learning

We now introduce the detailed procedures of off-line classifier learning as shown in Fig. V-1.

(a) *Data Source Preparation:* Before we start to conduct dynamic program analysis and profile the behavior of worm and benign executables, we need to collect a large number of such executables as the data source. This set of executables is labeled into two classes: worm executables and benign executables. The worms are obtained from the Web site *VX Heavens* (http://vx.netlux.org).

(b) *Collection Dataset – Dynamic Properties of Executables:* With the prepared data source, we now discuss how to collect the dataset, referred to as dynamic properties of executables. Recall that in order to accurately distinguish worm executables from benign ones, we need to collect data that can capture the fundamental behavior differences between them – the dynamic properties. One feasible and efficient method we choose is to execute the executables and trace the run-time system call sequences during their execution. However, executing worms might damage the host operating systems or even the driven of computer hardware. In order to solve this problem in our experiments, we set up virtual machines as our experimental test-bed. Then we launch each executable in our data source and record its system call trace during the execution on the virtual machine. The collection of the system call traces for each executable in our data source is referred to as a *dataset*. We split the dataset into two parts: the *training set* and the *test set*. With the training set, we will apply classification learning algorithms to learn the classifier. The concrete format and content of the classifier is determined by the adopted learning algorithms. With the test set, we will further evaluate the accuracy of the learned classifier on classification of new and unidentified executables.

(c) *Feature Extraction:* With the collection dataset consisting of system call trace of different executables, we extract all the system call sequence segments with a certain length. These segments are referred as *n*-gram, where the *n* is the length of the sequence, i.e., the number of system calls in one segment. These *n*-grams can represent the relative independent and

meaningful *action* taken during the program execution, or *program block* in the executables. We intend to use these *n*-grams to capture the behaviors of common worms and benign executables. Hence, these *n*-grams are the features for classifying worms and benign executables and each distinct *n*-gram represents a specific feature in our classification.

(d) *Classifier Learning:* From the features we extract from the training dataset, we need to learn a classifier that can distinguish between worms and benign executables. When we select the classification algorithm, we must consider both the accuracy of the learned classifier and the interpretability of the classifier. Some classifiers are easy to interpret and the classification (i.e., *decision rule* of worm detection) can be easily extracted from the classifier [38]. Then, the worm writers can use the *rules* to change the worm behavior and consequently evade detection, similar to the self-mutating worms that change themselves to defeat signature-based detection [62]. Thus, we need classifiers with very low interpretability. In our case, we consider two algorithms, *Naive Bayes*-based algorithm and *Support Vector Machine* (SVM) algorithm, and evaluate their performance. While Naive Bayes-based algorithm is simple and efficient in classifier learning, SVM is more accurate. More importantly, SVM learns a *black-box* classifier, which is hard for worm writers to interpret.

3) On-line Worm Detection

Having the learned classifier in the off-line process, we now describe how to use it to carry out on-line worm detection. In this process, we intend to automatically detect a new and unseen executable.

In particular, we follow the same procedure as in the off-line process, in which system call traces of an unknown executable are recorded and classification features (e.g., system call sequence segments with certain lengths) are extracted during its execution. Then, the

classification algorithm with the learned classifier is applied to classify the new executable, i.e., whether it belongs to the worm class or benign one.

In fact, the aforementioned worm detection actually depends on the accuracy of the classifier. In order to evaluate it, we use it to classify the executables in the *test set*. Since we know the class label of these executables, we can simply compare the classification results from the learned classifier with the pre-known labels. As such, the accuracy of our classifier can be measured.

In the following sections, we will present the major steps listed above, e.g., dataset collection, feature extraction, classifier learning, and on-line worm detection in detail, followed by experiment results.

3.b. Dataset Collection

In this section, we present the details on how we obtain the dataset, i.e., the dynamic program properties of executables in the form of system call traces.

1) Worm Execution with Virtual Machine

In order to obtain the run-time behaviors of worm and benign executables, we need to execute the benign executables as well as worms. As we mentioned earlier, since execution of worms might damage the operating system and even the driver code of host hardware, we set up virtual machines (VMs) [84] as the testbed. The VM we choose is *VMware* [85].

Even with VMs, two difficulties can still arise during data collection because of the worm execution. First, since worms can crash the operating system (OS) in the VM, then we might have to repeatedly re-install the OS. In order to avoid these tedious re-installations, we first install all necessary software for our experiments and store all of our worm executables on the VM, and then save the image file for that VM. Whenever the VM OS crashes, we can clone the

identical VM from the image file to continue our experiment. Second, it is difficult to obtain the system call traces from the VM after it crashes. In order to solve this problem, we set the *physical machine*, on which a VM is installed, as the network neighbor of the VM through the virtual network. Thus, during the execution of worms, the VM automatically outputs the system call trace to the physical machine. Although the physical machine can be attacked by the worms on the VM because of this virtual network, the physical machine is well protected by the dedicated host-based firewall and updated anti-virus software with very restricted access controls.

2) System Call Trace

Recall that we choose dynamic properties of executables to capture the executables' behavior and more accurately distinguish worms from benign executables more accurately. There are multiple dynamic program analysis methods [68, 69] that can be used to investigate the dynamic properties of executables.

The most popular methods are debugging and simulation. However, they have to be used manually with expertise to study the execution (behavior) of programs. In our case, they are not suitable for automatic analysis without humans' intervention. However, execution tracing is a good method for automatic analysis, which can automatically record run-time behavior of executables. Also, it is easy to analyze the trace using automatic analysis algorithms.

There are different ways to carry out execution tracing. In our case, we choose to trace system calls of worm and benign executables and use the trace as the source of classification (worm detection). The reasons for doing so is straightforward. Tracing all Microsoft Windows Application Programming Interface (API) functions can capture more details about the run-time behavior of executables. However, it increases OS resources consumption and interference with the execution of other programs, compared with tracing only system calls. The reason is that, the

number of system calls, 311 for all the Windows version together [86], and 293 for Linux 2.6 kernel [87], is significantly less than the number of APIs, over 76,000 for Windows version before Windows Vista [88], over 1000 for Linux [89]. Hence, we choose to trace only system calls and hence build a lightweight run-time worm detection.

3.c. Feature Extraction

Features are key elements for any anomaly detection or classification. In this section, we describe our method to extract and process the features that are used to learn the classifier and carry out worm detection.

1) N-gram from System Call Trace

System call traces of executables are the system call sequences (time series) of the execution, which contains the temporal information of program execution and thus the dynamic behavior information of the executables. In our system, we need to extract appropriate features that can capture common or similar temporal information hidden in the system call sequences of all worm executables, which is different from the temporal information hidden in the system call sequences by all benign executables.

The *n*-gram is a well-accepted and frequently adopted temporal feature in various areas of statistical natural language processing and genetic sequence analysis [90, 91]. It also fits our temporal analysis requirement. An *n*-gram is a subsequence of *n* items from a given sequence. For example, if a system call sequence is {*NtReplyWaitReceivePortEx, NtOpenKey, NtReadVirtualMemory, NtCreateEvent, NtQuerySystemInformation*}, then the 3-grams from this sequence are {*NtReplyWaitReceivePortEx, NtOpenKey, NtReadVirtualMemory*}, {*NtOpenKey, NtReadVirtualMemory, NtCreateEvent*}, and {*NtReadVirtualMemory, NtCreateEvent, NtQuerySystemInformation*}.

We use *n*-grams as the features in our system for the following reasons. Imagine the difference between one line of source code and one *block* of source code in a program. One line of code provides little meaningful information of a program, but one block of code usually represents a meaningful and self-contained small task in a program, which is the logical unit of programming. For a similar reason, one system call only provides very limited information about the behavior of an executable, whereas a segment of system calls might represent a meaningful and self-contained action taken during the program execution. Worm and benign executables have different behaviors, and this can be represented as the difference between their source code blocks, or the segments (i.e., *n*-grams) of their system calls. Hence, we use these system call segments, or the *n*-grams, as the features to classify worm and benign executables, which are shown to be very effective through our experiments, as described in Section 4.

2) Length of N-gram

One natural question is what length of *n*-gram is best for classifying worms from benign executables. On one hand, in order to capture the dynamic behavior of program execuation, *n* should be greater than 1. Otherwise, the extracted 1-gram list is actually the list of system calls invoked by the executables. This special case is the same as the method used by static program analysis to detect worms, which has no dynamic run-time information of executables.

On the other hand, *n* should not be very large for the following two reasons. First, if *n* is too large, it is very unlikely to find common or similar *n*-grams among different worm executables. In one extreme case, when *n* becomes very large, the *n*-grams are no longer small tasks. Instead, they become the entire execution of the executables. Because different worms cannot have the exact same sequence of system call invocations (otherwise they are the same worm), the classifier learning algorithms will fail to identify a common feature (i.e., the same system call invocations) among them, neither can the classifier learning algorithm to define a class that can

cover all the worms. In this case, the classification will not work. Second, if *n* is too large, the number of possible distinct *n*-grams, (311*n* for MS Windows since Windows has 311 system calls, 293*n* for Linux since Linux has 293 system calls) will be too large to be analyzed in practice. We will investigate the impact of *n*-gram length on worm detection in our experiments and report the results in Section 4.

3.d. Classifier Learning and Worm Detection

In this section, we present the details of the last step in the off-line classifier learning process (i.e., how to apply the classifier learning algorithm to learn the classifier after extracting the features). In particular, we use two classification algorithms: the *Naive Bayes* algorithm, which is a simple but popular learning algorithm, and the *Support Vector Machine (SVM)* algorithm, which is more powerful but more computationally expensive. We also present how to conduct on-line worm detection with each of the algorithms in detail.

1) Naive Bayes-based Classification and Worm Detection

The Naive Bayes classifier (also known as the Simple Bayes classifier) is a simple probabilistic classifier based on applying Bayes' theorem [74, 93]. In spite of its naive design, the Naive Bayes classifier may perform better than more sophisticated classifiers in some cases, and it can be trained very efficiently with a labeled training dataset. Nevertheless, in order to use the Naive Bayes classifier, one has to make the assumption that the features used in the classification occur independently.

In our case, we use the Naive Bayes classifier to calculate the likelihood that an executable is a worm executable (i.e., in worm class) and the likelihood that it is a benign one (i.e., in benign class). Then, the detection decision can be made, e.g. the executable belongs to the class having a larger likelihood.

(a) Off-line Classifier Learning

We represent each executable by an *m*-dimensional feature vector, $X = (x_1, x_2, \ldots, x_m)$, where *m* is the number of distinct *n*-grams in the dataset, x_i $(i=0, \cdots, m-1)$ is the *i*-th distinct *n*-gram $x_i = 1$ if x_i appears in the executable's system call trace, $x_i = 0$ otherwise. We have two classes, worm class C_w and benign class C_b. Given the feature vector, *X*, of an unknown executable, we need to predict the class to which *X* belongs. The prediction is made as follows. First, we calculate the likelihood that the executable belongs to different classes. Second, we make the decision based on the value of likelihood, e.g., the executable belongs to the class which has a larger likelihood for the given executable.

Actually, the off-line "classifier" learning process of the Naive Bayes algorithm is the preparation for the calculation of the above two likelihoods. Particularly, this preparation is the calculation of some statistical probabilities based on the training data. These probabilities are the posterior probability of each *n*-gram, say, x_i, conditioned on each class, C_w and C_b. Hence, the off-line "classifier" learning process in our Naive Bayes classification is actually the calculation of $P(x_i|C_j)$, $i = 1, \ldots, m$, and $j = w$ or b based on the training dataset. Remark that in some implementations, the classifier learning based on the Naive Bayes algorithm may conduct extra process, such as selection of features, cross-validation, but they are not the core procedures for the Naive Bayes algorithm.

(b) On-line Worm Detection

During the on-line worm detection, for each unknown executable, the feature vector *X* for that executable is built first. Then, we predict the class which *X* belongs based on a higher posterior probability, conditioned on *X*. That is, the Naive Bayes classifier assigns an unknown sample *X* to the class C_j if and only if

$$P(C_j \mid X) > P(C_{k|} \mid X) \quad j,k = w \quad \text{or} \quad b, j \neq k. \tag{V-1}$$

Based on Bayes theorem, $P(C_j \mid X)$ can be calculated by

$$P(C_j \mid X) = \frac{P(X \mid C_j)P(C_j)}{P(X)}. \quad \text{(V-2)}$$

In order to predict the class of X, we will calculate $P(X|C_j)P(C_j)$ for $j = m$ or b and consequently compare $P(C_w|X)$ and $P(C_b|X)$. Now we present how to calculate $P(X|C_j)P(C_j)$. First, if the class prior probabilities $P(C_w)$ and $P(C_b)$ are not known, then it is commonly assumed that the classes are equally likely, that is $P(C_w) = P(C_b)$. Otherwise, $P(C_j)$ can be estimated by the proportion of class C_j in the dataset. Second, for $P(X|C_j)$, as we assume the features are independent, $P(X|C_j)$ can be calculated by

$$P(X \mid C_j) = \prod_{i=1}^{m} P(X_i \mid C_j), \quad \text{(V-3)}$$

where $P(x_i|C_j)$ can be calculated during the off-line classifier learning process.

(c) Discussion

The Naive Bayes classifier is effective and efficient in many applications. The theoretical time complexity for learning a Naive Bayes classifier is $O(Nd)$, where N is the number of training examples and d is the dimensionality of the feature vectors. The complexity of classification for an unknown example (an unknown executable in our case) is only $O(d)$.

However, the Naive Bayes classifier has two limitations in our case. First, the classifier learned by Naive Bayes-based method can be used by the worm writer to make the worm detections less effective for new worms. The Bayes Naive classifier in our approach is actually a set of probabilities in which the *n*-grams appear in each class. Worm writer can directly use these information to make new worms similar to benign executables by either using or avoiding certain *n*-grams (system call sequences). Second, high accuracy of the Naive Bayes classifier is based on the assumption that the features are independent to each other. However, the *n*-grams in

the system call trace of an executable might not be independent in reality. In order to address these problems of Naive Bayes classifier, we will use the Support Vector Machine (SVM) in our worm detection as described in the following subsection.

2) Support Vector Machines-based Classification and Worm Detection

The Support Vector Machine (SVM) is a type of learning machine based on statistical learning theories [94, 95, 96]. The SVM-based classification includes two processes. One is classifier learning and the other is the classification. The classifier learning is to learn a classifier/model, using the training dataset. Then the learned classifier is used to determine/predict the class label of instances that are not contained in the training dataset. The SVM is a sophisticated and accurate classification algorithm. Although it is computationally expensive, its trained classifier is difficult to interpret. These silent features match our requirements for accurate worm detection and interpretation difficulty for worm writers.

(a) Off-line Classifier Learning

A typical SVM classifier learning problem is to label (classify) N training data $\{x_1, \ldots, x_N\}$ to a positive or negative class, $x_i \in R_d$ $(i = 1,\ldots ,N)$, where d is the dimensionality of the samples. Remark that the SVM algorithm can be extended to classification for more than two classes, but the two classes are the typical and basic cases. Our problem belongs to the classification of two classes. Thus, the classification result is $\{(x_1, y_1), \ldots , (x_N, y_N)\}$, $y_i \in \{-1,+1\}$. In our case, x_i is the feature vector built for the i-th executable in our dataset. That is, $x_i = \{x_{i,1}, \ldots , x_{i,d}\}$, where d is the number of distinct n-grams, $x_{i,j}$ $(j = 1, \ldots , d)$ is the j-th n-gram, $x_{i,j}=1$ if $x_{i,j}$ appears in the ith executable's system call trace, $x_{i,j} = 0$ otherwise. $y_i = +1$ means that x_i belongs to worm class, $y_i = +1$ means that x_i belongs to benign executable class. As we have a large number of features (n-gram), the dimensionality of the Euclidean space in our classification

problem is very large (upper bounded by *311n* depending on *n*-gram length). There are two cases for the SVM classifier learning problems; (i) the samples in the two classes are linearly separable; (ii) the samples in the two classes are not linearly separable. But case 2 holds for most real-world problems. In the SVM, in order to achieve an optimal classifier, the non-linear solvable problem in case (2) needs to be transformed to be a linear solvable problem in case (1) first. Then, the optimal classifier can be learned through linear optimization [93, 94]. In the following, we first present the algorithm for the simple case (case (1)), followed by the algorithm for case (2).

(i) Case 1: Classes are linearly separable

If the two classes are linearly separable, then we can find a hyperplane to separate the examples in two classes as shown on the right side of Fig. V-3. Examples that belong to different classes should be located on different sides of the hyperplane. The intent of the classifier learning is to obtain a hyperplane which can maximally separate the two classes.

Mathematically, if the two classes are linearly separable, then we can find a hyperplane $w \cdot x + b = 0$ with a vector w and an intercept b, that satisfies the following constraints:

$$w \cdot x_i + b \geq +1 \qquad \text{for} \quad y_i = +1, \tag{V-4}$$

$$w \cdot x_i - b \leq -1 \qquad \text{for} \quad y_i = -1, \tag{V-5}$$

or, equivalently

$$y_i (w \cdot x_i - b) - 1 \leq 0 \qquad \forall i. \tag{V-6}$$

Examples in the training set that satisfy the above equality are referred as support vectors. The support vectors define two hyperplanes, one going through the support vectors of the positive class and the other going through the support vectors of the negative class. The distance between these two hyperplanes defines a margin and this margin is maximized when the norm of the vector w ($\|w\|$) is minimized. When this margin is maximized, the hyperplane $w \cdot x + b = 0$

separates the two classes maximally, which in fact is the optimal classifier in SVM algorithm. The dual form of Formula (V-6) reveals that the above optimization actually maximizes the following function,

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{1} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j (X_i \cdot X_j) y_i y_j, \qquad \text{(V-7)}$$

subject to the constraint that $\alpha_i \geq 0$. The SVM algorithm can achieve the optimal classifier by finding out $\alpha_i \geq 0$ for each training sample x_i to maximize $W(\alpha)$.

(ii) Case 2: Classes are not linearly separable

In the above case, the optimization can be achieved for classes that are linearly separable. However, the real-world classification problems usually cannot be solved by the linear optimization algorithm. This case is illustrated as the left side of Fig. V-3, in which, there is no linear hyperplane (e.g., in this case, it is a straight line in 2-dimensional space) that can separate the examples in two classes (here shown with different colors). In other words, the classifier needed must be a curve, which is difficult to optimize.

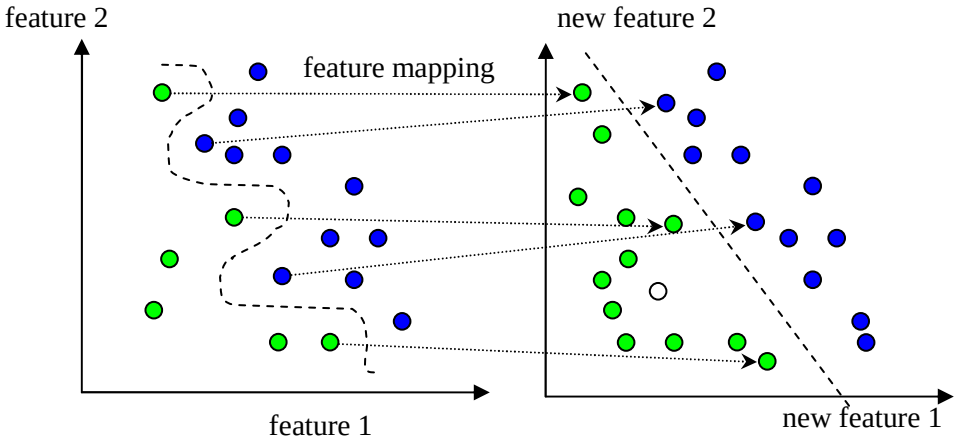


Fig. V-3. Basic Idea of Kernel Function in SVM

The SVM provides a solution to this problem by transforming the original feature space into some other, potentially high dimensional, Euclidean space. Then, the mapped examples in the training set can be linearly separable in the new space as demonstrated by the right side of Fig. V-3. This space transformation can be implemented by a *kernel function*,

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j), \quad \text{(V-8)}$$

where $\Phi(x_i)$ is the mapping from the original feature space to the new Euclidean space. We would only need to use $K(.)$ in the classifier training process with Equation (V-7), and would never need to explicitly know what Φ is. The SVM kernel function can be either linear or non-linear. Common non-linear kernel functions include Polynomial, Radial Basis Function (RBF), and Sigmoid among others.

(b) On-line Worm Detection

The on-line worm detection is the classification of new executables, using the SVM classification algorithm along with the optimal SVM classifier learned during the previously-discussed off-line learning process.

For an unknown executable (a worm or benign executable), its feature vector must be built first. The method is the same as the process aforementioned on the executables in the training set. That is, the system call trace during the execution is recorded, then the *n*-grams with certain value of *n* is extracted. After that, the feature vector, x_k, of this executable is formed from its trace, using the same method as in the off-line classifier learning process.

Recall that during the classifier learning process, the optimal hyperplane is found. Then, for a new example x_k shown as the small circle in Fig. V-3, the on-line classification is to checks on which side of the optimal hyperplane x_k is. Mathematically, the classification is conducted through signing a class to the executable by

$$C(x_k) = sign(w \cdot x_k - b), \quad \text{(V-9)}$$

where

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i . \qquad \text{(V-10)}$$

If $C(x_k)$ is positive, we predict the executable is a worm. Otherwise, we predict it as benign.

(c) Complexity of SVM

The classifier learning process of SVM is relatively time consuming because of the large volume of training set, high dimension of our feature space, complexity of classifier calculation and optimization. No matter what kernel function is used, if N is the number of training examples, N_s is the number of support vectors, and d is the dimension of the original feature vectors for the training examples, then the complexity upper bound of SVM classifier learning is $O(N_s^3 + N_s^2 N + N_s dN)$. However, the SVM classification process for each new executable is fast and involves only limited calculations. Its complexity is $O(MN_s)$, where M is the complexity of the kernel function operation. For Radio Basis Function kernels, M is $O(d)$.

(d) Black-Box Characteristics of the SVM Classifier

The classifier learned by the SVM can be easily used to carry out worm detection. However, the SVM classifier is difficult to interpret. The SVM classifier learning algorithm generates black-box models (classifiers) in the sense that they do not have the ability to explain in an understandable form [97, 98, 99]. Thus, from the SVM classifier, it is hard to extract decision rules comprehensible in the original problem domain, especially for the non-linear SVM, due to the feature space transformation introduced by kernel functions.

The above characteristic of SVM is a well-known limitation for the applications in which one needs to know the decision rules which can be mapped back to the physical entities in the original problem domain. However, this characteristic can help us prevent the worm writers from interpreting and learning from the classifier. We want to prevent the worm writers from

obtaining the signature of their worms or any benign executable. Otherwise, the worm writer can hide new worms accordingly as benign executables.

Besides the optimization algorithm used in SVM, the learning classifier also depends on the definition of input feature space, the selection of kernel function, the parameters of the kernel function, etc., which are unknown to worm writers. The worm writer does not know the following: the value of *n* of the *n*-gram used in the classifier, the mapping between *n*-grams and feature indices in the feature vector, the definition of the kernel function, the parameters of the kernel function, and the space transformation introduced by kernel function.

Hence, even if the worm writer knows that we use SVM and are able to get the classifier, it is hard for him to interpret the classifier to discovery the decision rule we used to distinguish between worms and benign executables. Thus, it is hard for him to change the worm behavior accordingly to evade our detection. Furthermore, we can protect the classification by mechanisms, such as encryption.

4. Performance Evaluation

In this section, we first present the experimental setup and metrics. Then we report on the results of our experiments.

4.a. Evaluation Methodology

In our experiments, we use 722 benign executables and 1589 worms in Microsoft Windows or DOS Portable Executable (PE) format as the data source, although our approach works for worm detection on other operating systems as well. We use this data source to obtain the generic worm classifier and further evaluate the trained classifier to detect worms. This set of executables are labeled into two classes: worms and benign executables. The set of worms obtained from the Web site *VX Heavens* (*http://vx.netlux.org*) have email worms, peer-to-peer (P2P) worms,

Instance Message (IM) worms, Internet Relay Chat (IRC) worms, and other non-classified worms. The benign executables in our experiments include Microsoft software, commercial software from other companies or free open source software. This diversity of executables enables us to obtain classifiers comprehensively that capture the behaviors of both different types of worms and benign executables. We use 80% of each class (worm and benign) as the training set to obtain the classifiers. We use the remaining 20% as the test set to evaluate accuracy of the classifiers, i.e., the performance of our detection approach.

We install MS Windows Professional 2000 with service pack 4 on our virtual machines. On these virtual machines, we launch each executable in our executable collection and use *strace for Windows NT* [99] to trace their system calls for 10 seconds. Recall that we trace the executables in the data set for longer time, then use a slide window to capture certain length trace for the classifier training. We found that using 10 second trace is enough to provide high detection accuracy. From the trace file of each executable, we extract the system call name sequences in the time order. Then we obtain the segment of system calls (i.e., the *n*-grams), given different value of *n* for each executable. After that, we build the vector inputs for the classification learning algorithms.

Recall that the classification in our worm detection problem is in a high dimensional space. There are a large number of dimensions/features which cannot be handled or handled efficiently by many data mining tools. The data mining tools we choose are Naive Bayes classification tools from University of Magdeburg in Germany [100] and *svm light* [101]. Both of the tools we selected are implemented in C language, and perform efficiently, especially for a high dimension classification problem. When we apply SVM algorithm with *svm light*, we choose Gaussian Radial Basis Function (Gaussian RBF), which has been proven to one of the effective kernels [73]. The distribution of features follows Gaussian distribution. Gaussian RBF is in the form of

$$K(x_i, x_j) = e^{-r\left\|x_i - x_j\right\|^2}, \quad \text{(V-11)}$$

which means (V-8) needs to be replaced by (V-11) in the classifier learning process and on-line worm detection process. The value of *r* is optimized through experiments and comparison.

In order to evaluate the performance of our classification for new worm detection, we can use two metrics, *Detection rate* (P_D) and *false positive rate* (P_F). In particular, the detection rate is defined as the probability that a worm is correctly classified. The false positive rate is defined as a benign executable classified mistakenly as a worm.

Table V-1. Detection Results for the Naive Bayes-Based Detection

n-gram length (*n*)	1	2	3	4	5	6
Detection Rate (P_D)	69.8%	81.4%	85.0%	90.9%	93.6%	96.4%
False Positive Rate (P_F)	33.2%	18.6%	11.5%	8.89%	6.67%	6.67%

Table V-2. Detection Results for the SVM-Based Detection

n-gram length (*n*)	1	2	3	4	5	6
Detection Rate (P_D)	89.7%	96.0%	97.73%	99.5%	99.5%	99.5%
False Positive Rate (P_F)	33.3%	18.75%	7.14%	4.44%	2.22%	2.22%

4.b. Experiment Results

In this subsection, we report on the performance of our worm detection approaches. The results of Naive Bayes and SVM-based worm detections in terms of *Detection Rate* and *False Positive Rate* under different *n*-gram length (*n*) are shown in Table V-1 and V-2, respectively.

(a) Effectiveness of Our Approaches

We conclude that our approaches of using both the Naive Bayes and SVM algorithms can correlate detect worm at a high detection rate and low false positive rate when the length of *n*-gram is of a reasonable value. For example, when the length of *n*-gram is 5, the detection based on the SVM algorithm achieves 99.5% detection rate and 2.22% false positive rate and the detection based on the Naive Bayes algorithm achieves 96.4% detection rate and 6.67% false positive rate, respectively.

From these tables, we also conclude that SVM-based detection performs better than Naive Bayes-based detection in terms of both detection rate and false positive rate. There are two reasons for this. First, the Naive Bayes classification assumes that features are independent, which might not be always true in real practice. Second, for the Naive Bayes-based classification, the calculation of the likelihood for classifying a new executable is based on the vectors of the training set executables in the feature space. Then, it predicts the class of the new executable simply based on the comparison of the likelihood. Differently, the SVM attempts to optimize the classifier (hyperplane) through finding the hyperplane that can maximally separate the two classes in the training set.

(b) Impacts of N-gram Length

Another important observation is the length of *n*-gram, i.e., the value of *n*, impacts the detection performance. When *n* increases from 1 to 4, the performance keeps increasing. When *n* further increases, the performance does not increase, or it only increases very little. The reason can be explained as follows. First, when $n = 1$, each *n*-gram only contains one system call and thus contains no dynamic system call sequence and executable's behavior information. Actually, this special case is the static program analysis, which only investigates the list of system calls used by the executables. Second, when *n* is larger, the *n*-grams contain a larger length of system

call sequence and thus obtain more dynamic behavior of the traced executables. Hence, the detection performance is better. This also demonstrates that our dynamic program analysis approach outperforms the traditional static program analysis-based approaches. From the previous observation on the length of *n*-gram, we conclude that certain *length* of *n*-gram is effective enough for worm detection. This length (value of *n*) can be learned through experiments: when the increase of *n* brings little detection performance gain, that *n* value is good enough and can be used in practice. This method is actually used for other *n*-gram-based data mining applications [91, 92]. Furthermore, for the efficiency of worm detection, the *n* value should not be very long, as we discuss in Section 3.

5. Summary

In this chapter, we studied the countermeasure based on the dynamic signature of worm executables. Specifically, we proposed a new worm detection approach based on mining the dynamic execution of programs. Our approach is capable of capturing the dynamic behavior of executables and providing efficient and accurate detection against both seen and unseen worms. Using a large number of real-world worm and benign executables, we ran executables on virtual machines and recorded run-time system call traces of these executables. We then applied two data mining classification algorithms to learn about classifiers off-line, which are subsequently used to carry out on-line worm detection. Our data clearly showed the effectiveness of our proposed approach in detection worms in terms of both very high detection rate and low false positive rate.

Our proposed approach has the following advantages. It is practical with low overhead during both classifier learning and run-time detection. Our approach does not rely on investigation for individual executable; rather, it examines the common dynamic properties of

executables. Therefore, it can automatically detect brand new worms and other unseen worms such as polymorphic worms. Furthermore, our approach attempts to build a black-box classifier which makes it difficult for the worm writers to interpret our detection.

CHAPTER VI

COUNTERMEASURE BASED ON CONTRADICTED OBJECTIVES

In this chapter, we focus on developing the countermeasure based on contradicted objectives to defend against worm attacks that change their patterns to circumvent the detection.

1. Overview

Generally speaking, a worm attacker (or propagator) has two objectives: One is to infect as many computers as possible. The other is to avoid being detected and punished by the defensive system. After infecting a number of computers without being detected, the worm attacker can remotely control the infected computers and use them as stepping stones to launch additional attacks [3, 4, 5, 6, 7, 8, 9, 10]. Recent studies show the existence of a black-market for trading/renting compromised computers (as "bots") for future attacks [9, 10], providing further economic incentives for worm attacks.

Unfortunately, most existing countermeasures make a tacit assumption that worms always propagate at the highest possible speed. Nonetheless, some newly developed worms contradict this assumption by intentionally reducing their propagation speed to detection. For example, the "Atak" worm [102] and the "self-stopping" worm [42] circumvent detection by hibernating (i.e., stop propagating) periodically. If a worm can successfully avoid (or delay) detection, it may eventually infect more computers, resulting in more damage to the Internet.

In order to address threats from these new kinds of worms, we formulate a new class of worms, called self-adaptive worms, in this chapter. These worms adapt their propagation schemes to defensive countermeasures, aiming to avoid or delay detection, and ultimately infecting more computers. We propose and evaluate countermeasures against self-adaptive worms. Specifically, we partition self-adaptive worms into two classes. *Static self-adaptive*

worms are those that intelligently select a propagation speed at the time of attack launch but nevertheless maintain a constant speed during the attack session. For a *dynamic self-adaptive worm*, its propagation speed may vary during the attack session. Remark that the *camouflaging worm* studied in Chapter III is a special case of dynamic self-adaptive worm by adopting feedback loop-control to manipulate a traffic pattern.

To develop proper countermeasures, we introduce a game-theoretic formulation to model the interaction between the worm propagator and the defender. We show that an effective integration of multiple countermeasure schemes (e.g., worm detection and forensics analysis) is critical for defending against self-adaptive worms by enforcing the worm attack to choose between the objectives.

In the following, we will first present models for worms and defensive schemes. We then introduce a baseline system where a static self-adaptive worm freely propagates without defensive countermeasures and introduce a game-theoretic formulation of the system to model the interaction between self-adaptive worms and countermeasures. Based on the game-theoretic formulation, we then present our countermeasures against static and dynamic self-adaptive worms.

2. Models

In this section, we present models for worms and defensive schemes. In particular, we start with the propagation model for traditional worms and then formally define a propagation model for self-adaptive worms. After that, we present our models for defensive countermeasures.

2.a. Worms

1) Traditional Worms

Let us first consider traditional worms investigated in previous work [2]. Generally speaking, a traditional worm behaves similar to biological viruses in terms of its greedy self-propagating nature. Worm propagation on the Internet is an iterative process that usually starts with a computer, known as the worm propagator. The worm propagator conducts a network propagation scan to identify vulnerable computers on the Internet, and then infects these computers by remotely exploiting the vulnerabilities to obtain access privileges. Once a computer is infected by the worm, the computer will then recursively start propagating the worm to other computers on the Internet.

In order for a worm to propagate itself on the Internet, it must be capable of identifying computers with certain vulnerabilities. Given the complex topology of vulnerable computers on the Internet, such identification can be hardly optimal in practice. A commonly used identification strategy is Pure Random Scan (PRS) [1, 2, 10, 16], in which each worm-infected computer randomly scans IP addresses to identify vulnerable computers. To improve the performance of the PRS approach, work has been done, which enables worm to carry a *hit-list*, containing certain addresses of pre-known vulnerable computers [18]. Note that the length of the hit-list is limited by the size of the worm. Thus, this approach may not be able to support the wide propagation of a worm. For the sake of simplicity, we only consider the PRS propagation mechanism in this chapter.

Most previous studies [1, 2, 10, 16] make a tacit *maximum speed assumption* on worm propagation: A worm-infected computer always scans the network with the maximum possible speed. Formally, let S be the maximum number of scans that an infected computer can perform in a unit of time. Let $p(t)$ be the percentage of S that a worm actually scans at time t. That is, the

number of scans that an infected computer actually performs at time t is $p(t) \cdot S$. We refer to $p(t)$ as the propagation growth rate at time t. Due to the maximum speed assumption, the traditional worms have $p(t) = 1$ for all t.

2) Self-Adaptive Worms

With defensive systems in place nowadays, worms have consequently evolved and become more sophisticated than the traditional worms mentioned above. In particular, some worms deliberately reduce their propagation speed to avoid detection [46, 102]. In this chapter, we propose to deal with these new, smarter worms. Specifically, we remove the maximum speed assumption, and consider self-adaptive worms that manipulate their propagation growth rate in order to avoid or delay detection. Formally, a self-adaptive worm is a generalization of traditional worms with $p(t) \leq 1$.

In an ideal situation, when $p(t)$ is very small (i.e., $p(t) \approx 0$), a self-adaptive worm may propagate forever without being detected. In practice, however, it only makes sense for a worm to propagate for a finite amount of time. Thus, we make a finite propagation assumption that a worm will only propagate for a finite (yet very long) amount of time t_E. This finite propagation assumption is reasonable in practice because the vulnerable computers will eventually be fixed and the worm will be detected. Based on the finite propagation assumption, the objective of worm on propagation becomes to infect as many computers as possible by time t_E.

Algorithm VI.1 Propagation of self-adaptive worms

Require: Maximum scan rate S, Propagation growth rate $p(t)$, and finite time t_E

1: ***for all*** *$t = 0$ to t_E* ***do***

2: Current time is t

3: Determine the propagation growth rate p(t);

4: Launch p(t)·S scans to selected targets (e.g., via PRS) in this unit of time;

5: end for

Algorithm VI.1 shows the pseudo-code of a self-adaptive worm. As we can see, a self-adaptive worm can either use a constant $p(t)$ for the duration of worm propagation, or deliberately change $p(t)$ over time during the propagation. We consider both cases in this chapter. In particular, we call the self-adaptive worms with constant $p(t)$ as static *self-adaptive worms*. If a self-adaptive worm has $p(t)$ changed over time t, we call it *dynamic self-adaptive worms*. For static self-adaptive worms, we use p to denote the constant value of $p(t)$.

Note that each kind of worm has its own advantages and disadvantages. Static self-adaptive worms are easy to implement while the dynamic ones require each infected computer to compute the amount of time elapsed since the start of propagation and determine $p(t)$ correspondingly. Nonetheless, dynamic self-adaptive worms may outperform the static ones in terms of infecting computers and avoiding detection. The "Atak" worm [102] and the "self-stopping" worm [43] are special cases of dynamic self-adaptive worms, as their propagation growth rates are changing between 0 and 1 over time.

2.b. Countermeasures

Various countermeasure schemes have been proposed to defend against worm attacks. We consider two types of defensive schemes in this chapter: One is the *worm detection* scheme, which focuses on the detection of propagating worms on the Internet. Once a propagating worm is detected, many actions can be taken to stop or slow down worm propagation: For example,

patches can be released to fix the vulnerability; worm scan traffic can be throttled and filtered; and infected computers can be identified and quarantined [10].

The other type of scheme we consider is *trace-back*, which aims to identify the origin of worm propagation, such that appropriate legal steps can be taken to punish the worm propagator. As we will show in the chapter, if successfully deployed, this scheme can prevent worm propagators from launching attacks.

There has been much work on specific algorithms of detection and trace-back schemes. Please note that we do not intend to study the performance of these algorithms in this chapter. Rather, our objective is to analyze the effectiveness of the entire classes of detection and trace-back schemes. For this purpose, we will introduce models for detection and trace-back schemes. These models are representative of many algorithms that have been developed but still simple enough to enable our quantitative analysis. We will also propose a framework that integrates detection and trace-back schemes.

1) Detection Schemes

A typical defense system with detection scheme usually is based on the ITM system which consists of a number of monitors and a data center. Each monitor is responsible for monitoring suspicious traffic (e.g., scan to unoccupied IP addresses or ports) targeted to a range of IP addresses and reporting the collected traffic logs to the data center periodically. The data center issues alerts of worm propagation by analyzing the characteristics of traffic recorded in the logs. In this chapter, we consider a simple detection mechanism of using average traffic volume in the *threshold-based scheme* [31]. With this scheme, the data center issues an alert if and only if the average volume of traffic collected in a given time period is larger than a pre-determined threshold T_R. Note that the threshold T_R must be carefully chosen for the detection scheme to be effective: In particular, it must minimize both false negative rate (i.e., the probability that an

ongoing worm attack is not reported) and false positive rate (i.e., the probability that an alarm is triggered when no worm is propagating). The proper selection of T_R will be further discussed in Section 3.

2) Trace-back Schemes

A trace-back scheme typically works as follows: (some of the) routers in the system monitor all traffic transmitted through the routers and record traffic logs in some network storage servers. When a "trace-back" order is given, the recorded information is analyzed to determine the origin of worm propagation [104, 105]. In order to successfully identify the worm propagator, the system must be capable of monitoring and recording traffic for a substantial amount of time. In particular, we use t_B to denote the maximum length of time interval during which all traffic information can be recorded in the storage servers.

Trace-back schemes cannot be precise in many real systems. Usually, the trace-back scheme reports a set of "suspects", rather than one computer, that could be the origin of the worm propagation. Then, law enforcement needs to take other means to investigate the suspects and capture the original worm propagator. To be effective, the set of suspects cannot be too large. Thus, we assume that in order to identify the worm propagator, it is required (by law enforcement) that the size of suspects set is no more than m ($m \geq 1$).

3) Integration of Threshold-Based and Trace-Back Schemes

We now introduce a defensive framework to integrate the threshold-based and trace-back schemes. The framework consists of a control center processing reports from numerous monitors as well as forensic support (storage) servers which are distributed across the Internet. Once the control center detects a propagating worm, it issues an order to initiate the trace-back process by collecting network traces from the forensic support servers. We assume that multiple sub-networks collaborate with each other by sharing the stored forensic data to jointly locate the

worm propagator through post-mortem analysis [106]. This defense framework will be further extended and discussed in Section 3 to integrate the new spectrum-based scheme we will propose in the chapter.

The proposed defense framework can be deployed using existing commercial products. For example, the *sinkhole* feature of Cisco's Private Internet Exchange (PIX) Firewall can be readily used by the distributed monitors to collect anomaly traffic such as illegal scans to IP addresses not occupied by real computers or other devices; Cisco's Netflow tool can be used to analyze traffic logs for forensic analysis; and Cisco's Security Management Solution (SIMS) or Arbor Network's Peakflow can be deployed on the control center [107, 108] to process the collected anomaly traffic.

3. A Baseline System

In this section, we analyze a baseline system in which a static self-adaptive worm freely propagates until time $t_E = \infty$ without any defensive countermeasure. This analysis forms the basis for us to analyze much more complicated systems, in which the worm may be dynamic self-adaptive, the maximum propagation time t_E is limited, and various defense schemes are deployed.

Let *f(t)* be the number of infected computers in the baseline system at time *t*. Without loss of generality, we assume that the following initial condition holds:

$$f(0) = 1. \tag{VI-1}$$

We are interested in the relationship between *f(t)* and other system parameters. To derive this relationship, we take an approach similar to the analysis of traditional worms with the simple epidemic model [45]. First, we have

$$f(t + \Delta t) = f(t) + X(t, \Delta t), \tag{VI-2}$$

where $X(t, \Delta t)$ is the number of computers infected during time internal $(t, t+\Delta t]$. $X(t, \Delta t)$ can be estimated as follows:

$X(t, \Delta t)$ = (Number of worm scans in $(t, t + \Delta t]$) · (Success rate of each scan). (VI-3)

Note that when $\Delta t \to 0$, the number of scans made during $(t, t + \Delta t]$ is equal to $S \cdot p \cdot f(t)$. Let V be the total number of IP addresses and N be the total number of vulnerable computers. At time t, the number of computers that are vulnerable to infection is $N - f(t)$. Then, the success rate of a scan is $(N - f(t)/V)$. Due to Formula (VI.3), we have

$$X(t, \Delta t) = S \cdot p \cdot f(t) \frac{N - f(t)}{V}. \quad \text{(VI-4)}$$

Substituting Formula (VI.4) into (VI.2), with some mathematical manipulation, we have

$$\frac{df(t)}{dt} = \lim_{\Delta t \to 0} \frac{f(t + \Delta t) - f(t)}{\Delta t} = \beta \cdot f(t) \cdot p \cdot (N - f(t)), \quad \text{(VI-5)}$$

where $\beta = S/V$ is called pair-wise propagation rate [46].

As we can see, (VI-5) is a differential equation of $f(t)$ in terms of system parameters S, V, N, and p. With the initial condition $f(0) = 1$, the equation can be easily solved using *Laplace* transform [109]. The solution is as follows:

$$f(t) = \frac{N \cdot e^{\beta \cdot p \cdot N \cdot t}}{e^{\beta \cdot p \cdot N \cdot t} + N - 1} \approx \frac{N \cdot e^{\beta \cdot p \cdot N \cdot t}}{e^{\beta \cdot p \cdot N \cdot t} + N}. \quad \text{(VI-6)}$$

Based on (VI-6), we would like to make the following remarks:

− $f(t)$ is an increasing function of t. Also, $f(t)$ increases when β, N, or p increases.

− When t is sufficiently small such that $e^{pNt} << N$, we have

$$f(t) = e^{\beta \cdot p \cdot N \cdot t}. \quad \text{(VI-7)}$$

That is, when a worm is in its initial propagation phase, the number of infected computers increases exponentially over time t.

− On the other hand, when t is sufficiently large,

$$f(t) = N. \quad \text{(VI-8)}$$

This indicates that when no defense system exists, eventually all vulnerable computers will be infected.

− Except for a new parameter "*p*", our result in (VI-5) is identical to the result in [10]. We nevertheless present the derivation process in this chapter to help our readers understand the physical meaning of the equation and its solution.

− Consider the extension of our baseline system to include the detection scheme. Let t_D be the time of detection. Then, (VI-6) will correctly represent the number of infected computers as long as $t \leq t_D$.

− While we derive *f*(*t*) for static self-adaptive worms, the derivation can be useful for the dynamic ones as well. From the derivation process, if we replace *p* by *p*(*t*) in (VI-4), the differential equation (VI-5) still holds. That is, (VI-5) can be used to describe dynamic self-adaptive worms as well. Unfortunately, the solution process used in (VI-6) requires that *p* be constant, and thus cannot be directly applied to dynamic self-adaptive worms.

4. Game-theoretic Formulation

We now consider the case in which both parties, the worm propagator and the defender, appear in the system. In this case, the number of infected computers will depend not only on the strategy of worm propagation (e.g., propagation growth rate *p(t)*), but on the defensive strategy and the interaction between the two parties as well. In particular, since one party may adapt to the strategy change of another party, the outcome of worm propagation is determined by the stable state where neither party can benefit by changing its strategy unilaterally. This state is referred to as the *Nash equilibrium* of the game between the worm propagator and the defender [109]. Our focus in the following section is to analyze the optimal strategies that constitute the Nash

equilibrium, when different combinations of self-adaptive worm and defensive schemes are present in the system. In order to do so, we first formulate the game model, and then present the strategy sets and utility functions of the two parties. The game-theoretic formulation introduced in this section will form the basis for the specific analysis of systems with static and dynamic self-adaptive worms in the next two sections.

4.a. Game

We formulate the system as a two-player uncooperative game. The worm propagator and the defender are the two players in the game. Each player P_i ($i \in \{1, 2\}$) has a strategy set S_i and a utility function $u_i : S_1 \times S_2 \rightarrow \acute{R}$ which we will introduce in the latter part of this section. The game is uncooperative in that the two players are in opposition and are unlikely to make any binding agreement when choosing their strategies [109]. As in many security studies, we make a conservative assumption that the worm propagator has full knowledge of the strategy taken by the defender. Nonetheless, the defender has no knowledge about the worm propagator's strategy.

We assume that both players are rational, in that each player P_i always chooses the strategy that maximizes its utility function u_i. The Nash equilibrium is a combination of strategies $\{s_1, s_2\}$ ($s_1 \in S_1, s_2 \in S_2$), such that $\forall s'_1 \in S_1, s'_2 \in S_2$,

$$u_1(s_1, s_2) \geq u_1(s'_1, s'_2), \tag{VI-9}$$

and

$$u_2(s_1, s_2) \geq u_2(s'_1, s'_2). \tag{VI-10}$$

As we can see, the Nash equilibrium represents a stable state because when equilibrium is reached, no player has an incentive to deviate from the chosen strategy (i.e., s_1 or s_2) unilaterally. Thus, we can evaluate the outcome of worm propagation based on the Nash equilibrium of the game.

4.b. Strategies

We now consider the strategy sets of the two parties. The strategy of the worm propagator is to determine the propagation growth rate p. Recall that as we mentioned in Section 1, the worm propagator can choose to either use a constant propagation growth rate p or to vary p over time t. Formally, the strategy set S^s_A of a static self-adaptive worm contains all possible values of p in [0, 1]. The strategy set S^d_A of a dynamic self-adaptive worm contains all possible functions $p(\cdot)$ that map time t in $[0, t_E]$ to a real number in [0, 1].

The strategy of the defender is to determine the parameters for countermeasures. Recall that as we mentioned in Section 2, we consider two kinds of countermeasures: threshold-based (i.e., worm detection) and trace-back (i.e., forensic analysis). Thus, the parameters include T_R for the threshold-based scheme, and t_B and m for the forensics analysis scheme. Since the trace-back parameters are determined by capacity of the defensive system and the trace-back algorithm [104, 107], we assume that the defender cannot change t_B or m. Thus, in our system model, the strategy of the defender is to determine the detection threshold T_R. Formally, the strategy set S_D of the defender contains all possible values of $T_R \geq 0$.

4.c. Utility Functions

The utility function $u_i(s_A, s_D)$ measures the benefit (or loss when $u_i < 0$) gained by Player P_i when a set of strategies s_A, s_D are chosen by the two players respectively. The utility function depends on the objectives of P_i. The worm propagator has two objectives. One is to maximize the number of infected computers. The other is to avoid being traced back and punished for its malicious actions. Although different worm propagators may have different priorities for these two objectives, it is commonly believed that most worm propagators on the Internet consider the

penalty of being traced back to be substantially more than the benefits gained from worm propagation [104, 105]. Thus, for the sake of simplicity, we assume that a worm propagator will suffer infinite loss if the probability of being traced back is more than 50%, but it will suffer no loss from forensic analysis otherwise. In Section 7, we will extend our results to the more general case in which loss of worm propagator from forensic analysis is a function of the success probability of trace-back.

Formally, the utility function of the worm propagator, denoted by u_A, is as follows:

$$u_A(s_A, s_D) = \begin{cases} -\infty, & \textit{If trace back with probability of more than } 50\%; \\ f(t_D), & \textit{otherwise.} \end{cases} \tag{VI-11}$$

where s_A and s_D are the strategies of the worm propagator and the defender, respectively, and t_D is the time when the worm is detected.

The defender also has two objectives. One is to minimize the number of infected computers. The other is to minimize the *false positive rate*, which is the probability that an alarm is falsely triggered when there is no worm propagation on the Internet. In our system model, we assume that the false positive rate Λ must be lower than a pre-determined threshold δ.

Formally, the utility function of the defender, denoted by u_D, is as follows:

$$u_D(s_A, s_D) = \begin{cases} -\infty, & \Lambda > \delta; \\ f(t_D), & \textit{otherwise.} \end{cases} \tag{VI-12}$$

In the following two sections, we will derive the Nash equilibrium of the game based on the strategy sets and the utility function of the two players.

5. Defense Against Static Self-Adaptive Worms

In this section, we consider a system with only traditional worms (p = 1) and static self-adaptive worms (constant p in [0, 1)). We first show that the threshold-based scheme, by itself, is

ineffective against static self-adaptive worms. After that, we demonstrate that an integration of the threshold-based and trace-back schemes can effectively defend against static self-adaptive worms.

5.a. Threshold-Based Scheme

We now show that if the defender only uses the threshold-based scheme, the game will reach Nash equilibrium in the state where the worm propagator cannot be detected before time t_E, and is capable of compromising a large number of computers.

Theorem VI-1. When the worm propagator propagates a static self-adaptive worm in the system and the defender uses threshold-based scheme only, the Nash equilibrium of the game is: The defender chooses $T_R = T_R^0$ where T_R^0 is the maximum value to satisfy $\Lambda \leq \delta$. The worm propagtor chooses $p = p_E$ such that $f(t_E) \cdot p_E = T_R^0$.

Proof: We show the correctness of the specified Nash equilibrium by proving that no player can benefit by changing its strategy unilaterally. Apparently, the defender cannot benefit by either increasing or decreasing T_R unilaterally because doing so will either keep the same u_D or reduce it to $-\infty$.

For the worm propagator, the current utility function is $u_A = f(t_E)$. Suppose that it changes the propagation growth rate to p_1. Let the new function of the number of infected computers be $f_1(\cdot)$. When $p_1 > p_E$, the worm will be detected at time $t_1 < t_E$ where $f_1(t_1) \cdot p_1 = T_R^0$. Since $p_1 > p_E$, we have $f_1(t_1) < f(t_E)$. Thus, the worm cannot benefit by changing to p_1 unilaterally. When $p_1 < p_E$, the number of infected computers at the time of detection is at most $f_1(t_E) < f(t_E) = u_A$. Thus, the worm cannot benefit by changing to p_1 unilaterally either.

We now illustrate the results of the theorem with practical examples. In particular, we set the system parameters as follows: N = 350,000 (the number of computers infected by the “Code-

Red" worm) [1], $V = 4 \times 10^9$ (i.e., the number of IP addresses in IPv4), S = 358 scans/second (the estimated value for "Code-Red" worm [1], δ = 3%, and t_E = 5 days. Based on the system settings [52], we compute T_R^0= 60,000 scans/minute. Due to the theorem, the optimal strategy for the worm propagator is to set p = 0.15. As such, the number of infected computers after t_E (5 days) is 71,400, or 20.4% of total vulnerable computers. This is a significant number that can cause substantial damage (a real-world worm that infected about 70,000 computers, the Slammer worm, resulted in about one billion dollars damage [2]. Thus, the threshold-based scheme by itself is ineffective against static self-adaptive worms.

As we can see from the theorem, when the threshold-based scheme is the only available defensive measure, the worm propagator can always reduce p to delay the detection until t_E. Thus, in order to defend against static self-adaptive worms, we have to introduce a countermeasure that prevents the worm propagator from reducing p to p_E. This motivates us to integrate the threshold-based scheme with the trace-back scheme. As we will show below, the trace-back scheme prevents the worm propagator from doing so because with a low propagation growth rate p, the worm propagator increases the chance of being traced back after detection.

5.b. Threshold-Based and Trace-Back Schemes

We now show that integration of the threshold-based and trace-back schemes can effectively defend against worm propagation. In particular, we have the following theorem. Recall that T_R^0 is the maximum value to satisfy $\Lambda \leq \delta$ and p_E satisfies $f(t_E) \cdot p_E = T_R^0$.

Theorem VI-2. When the worm propagator propagates a static self-adaptive worm in the system and the defender uses an integration of the threshold-based and trace-back schemes, the Nash equilibrium of the game is as follows:

− When

$$t_B \geq t_E(1-\frac{1}{\log T_R^0}\log\frac{m\cdot N}{N-m}) \approx t_E(1-\frac{\log m}{\log T_R^0}), \quad \text{(VI-13)}$$

the worm propagator chooses not to propagate the worm (i.e., $p = 0$). The defender chooses $T_R = T_R^0$.

− Otherwise, the worm propagator chooses $p = p_E$. The defender chooses $T_R = T_R^0$.

Proof*:* We prove the theorem by showing that no player can benefit by unilaterally changing its strategy. We first consider the case where $t_B \geq t_E(1 - \log m/\log T_R^0)$. Apparently, the defender already reaches the maximum possible $u_D = 0$ and cannot benefit by changing its strategy. For the worm propagator, suppose that it changes the propagation strategy to $p = p_1 > 0$. Consider $f(t_D - t_B)$, the number of infected computers at time $t_D - t_B$. Let $f_E(t)$ be the function of the number of infected computers when $p = p_E$. We have

$$f(t_D - t_B) < f_E(t_E - t_B) = \frac{N(e^{\beta\cdot p_E N t_E})^{1-\frac{t_B}{t_E}}}{(e^{\beta\cdot p_E N t_E})^{1-\frac{t_B}{t_E}} + N}. \quad \text{(VI-14)}$$

Since $t_B/t_E \geq (1 - \log m/\log T_R^0)$, with some mathematical manipulation, we have

$$f(t_D - t_B) < f_E(t_E - t_B) \leq m. \quad \text{(VI-15)}$$

As such, if the worm propagator changes its strategy to $p > 0$, the defender can always use the forensic analysis scheme to trace-back to the worm propagator with probability of at least 50%. That is, u_A will become $-\infty$. Thus, if $t_B \geq t_E(1 - \log m/\log T_R^0)$, the worm propagator will not change its strategy unilaterally.

When $t_B < t_E(1 - \log m/\log T_R^0)$, the game is exactly the same as the one discussed in Theorem VI-1, and thus follows the same Nash equilibrium.

As we can see from the theorem, there are two possible outcomes of worm propagation:

Outcome 1. If the trace-back interval t_B is longer than the threshold $t_E(1 - \log m/\log T_R^0)$, the threats posed by the trace-back scheme will force the worm propagator to not propagate the worm at all.

Outcome 2. If the trace-back interval is lower than the threshold, however, the worm will propagate in the same way as we discussed in Section 2, and infect a large number of computers before being detected.

We now analyze which outcome is likely to occur in practice based on practical examples. In particular, we would like to demonstrate that the derived lower bound on t_B in *Outcome 1* is reasonable in many systems: We use the same system setting as the one specified in Section 5.a. In addition, we set $m = 0.002 \cdot N$. Due to the theorem, no worm infection will occur if the trace-back interval t_B is more than *1.81* days. We argue that this is a reasonable trace-back interval for practical systems: Based on the real-world estimation of trace-back cost [103], the cost of realizing a trace-back interval of 1.81 days is approximately $216,000 per Internet service provider (ISP). Compared with the maintenance cost of ISP, the cost of trace-back is fairly moderate and acceptable in practice. Thus, an integration of the threshold-based and trace-back schemes can effectively defend again static self-adaptive worms as well as traditional worms.

The basic idea of the theorem can be stated as follows: With both the threshold-based and trace-back schemes in place, if the worm propagator chooses a larger p, it will be detected earlier, and the number of infected computers at time $t_D - t_B$ will be smaller. If the worm propagator chooses a smaller p to delay the detection until t_E, the worm will propagate slower and the number of infected computers at time $t_E - t_B$ will still be very small. If the trace-back interval t_B exceeds a threshold such that $f(t_D - t_B) \leq m$ in both cases, then the worm propagator will be forced not to propagate the worm because, otherwise, it will always be traced back and receive $u_A = -\infty$.

6. Defense Against Dynamic Self-Adaptive Worms

In this section, we consider a system with a dynamic self-adaptive worm, which changes its propagation growth rate $p(t)$ over time t to better adapt to the countermeasures. We first show that the integration of threshold-based and trace-back schemes are no longer effective against dynamic self-adaptive worms. After that, we introduce a new defensive scheme, called the *spectrum-based scheme*. We demonstrate that an integration of all three schemes can effectively defend against dynamic self-adaptive worms.

6.a. Threshold-Based and Trace-Back Schemes

We now show that the integration of threshold-based and trace-back schemes is ineffective against dynamic self-adaptive worms. In particular, we have the following theorem.

Theorem VI-3. When the worm propagator propagates a dynamic self-adaptive worm in the system and the defender uses an integration of threshold-based and trace-back schemes, the Nash equilibrium of the game is as follows:

− When $t_B \geq t_E - \log(m)/(N\cdot\beta) \approx t_E$, the worm propagator chooses not to propagate the worm (i.e., $p(t) \equiv 0$). The defender chooses $T_R = T_R^0$.

− Otherwise, the worm propagator chooses $p(t) = \min(1, T_R^0/f(t))$ for every t in $[0, t_E]$. The defender chooses $T_R = T_R^0$.

Proof: We first consider the case where $t_B \geq t_E - \log m/(N\cdot\beta)$. In this case, the proof of Nash equilibrium is similar to that of Theorem VI-2. Thus, we only demonstrate why the lower bound on t_B changes to $t_E - \log m/(N\cdot\beta) \approx t_E$. Consider the case where the worm propagator adopts a strategy as follows:

(i) First, the worm propagator uses $p(t) = \min(1, T_R^0/f(t))$ to infect m computers as soon as possible, say at time t_A (i.e., $f(t_A) = m$).

(ii) After that, the worm propagator chooses $p(t) = 0$.

As we can see, since $m << N$, the worm will not be detected before t_A. Thus, the worm propagator cannot be traced back as long as $t_A < t_E - t_B$. As such, in order to force the worm propagator not to propagate the worm, there must be $f(t_E - t_B) \leq m$ for the above strategy. That is,

$$\frac{N \cdot e^{\beta \cdot N(t_E - t_B)}}{e^{\beta \cdot N(t_E - t_B)} + N} \leq m. \tag{VI-16}$$

With some mathematical manipulation, we have

$$t_B \geq t_E - \frac{1}{\beta \cdot N} \log \frac{N \cdot m}{N - m} \approx t_E - \frac{\log m}{\beta \cdot N} = t_E. \tag{VI-17}$$

Thus, a necessary condition to force the worm propagator not to propagate the worm is $t_B \geq t_E - (\log m)/(N \cdot \beta) \approx t_E$.

We now consider the case where $t_B < t_E - \log m/(N \cdot \beta)$. In particular, we prove the correctness of the Nash equilibrium specified in the theorem by showing that no player can benefit by unilaterally changing its strategy. As we have shown in Theorem VI-2, the defender cannot benefit by deviating from $T_R = T_R^0$. For the worm propagator, suppose that it uses a different propagation growth rate function $p_1(t)$. In order for the worm propagator to benefit from the strategy change, there must exist t_1 in $[0, t_E]$ such that $p_1(t_1) > p(t_1) = \min(1, T_R^0/f(t))$. Nevertheless, the worm will then be detected at time t_1 due to the threshold-based scheme, resulting in a reduced u_A. Thus, no player can benefit by changing its strategy unilaterally from the equilibrium specified in the theorem.

As we can see from the theorem, the threats posed by the trace-back scheme are significantly weakened when the worm is dynamically self-adaptive. As such, the possible outcomes of worm propagation become:

Outcome 1. When the trace-back interval exceeds a very large threshold $t_E - \log m/(N \cdot \beta) \approx t_E$, the worm propagator will be forced not to propagate the worm.

Outcome 2. When the trace-back interval is lower than the threshold, however, the worm will propagate to more computers than what a static self-adaptive worm can infect in a system with the threshold-based scheme only.

We now analyze which outcome is likely to occur in practice based on practical examples. In particular, we demonstrate that the derived lower bound on trace-back interval t_B in *Outcome 1* is unachievable in many practical systems: Based on our system setting used in Sections 5.a and 5.b, no worm propagation will occur if and only if the trace-back interval is more than 4.8 days (i.e., $t_B \approx 4.8$ days). Based on the estimate of trace-back cost [103], in order to eliminate worm propagation, the cost of the trace-back scheme would be at least $2,430,000 per ISP, which is too high for the maintenance cost of an ISP in practice. Thus, the lower bound on t_B derived in the theorem is unachievable in practice. As such, an integration of the threshold-based and trace-back schemes cannot effectively defend again dynamic self-adaptive worms.

A critical observation from Theorem VI-3 is that in order to effectively defend against dynamic self-adaptive worms, the defender has to prevent the worm from rapidly propagating itself at the initial stage of worm propagation (i.e., before t_A where $f(t_A) = m$). Otherwise, the worm will quickly propagate to m computers before t_A, and then carefully choose $p(t)$ for $t > t_A$ to delay the detection until t_A+t_B, which makes the trace-back scheme useless. Since the threshold-based scheme is ineffective against self-adaptive worms by itself, the defender cannot eliminate worm propagation. This observation motivates us to propose the spectrum-based scheme, which prevents a worm from using high propagation growth rate at the initial stage of propagation.

6.b. Spectrum-Based Scheme

In the following, we introduce a spectrum-based detection scheme to restrict the propagation growth rate of a worm at the initial stage of propagation. Note that if a worm adopts a high propagation growth rate (e.g., $p(t) = 1$) at the beginning of propagation, the worm-scan traffic will exhibit a significant pattern (i.e., trend of exponential increase) when compared with the network background traffic. The objective of spectrum-based detection is to extract such a pattern (as signal) from the normal network traffic (as noise). The idea of using spectrum-based approaches to identify signal from noise has been widely used in the literature of signal processing [57], and has been shown to be capable of differentiating signal from noise even when the signal-to-noise ratio is low.

The objective of spectrum-based detection is to identify the (approximate) exponential growth of worm scan traffic from background traffic, which can be considered as white noise. In order to do so, we use *discrete Fourier* transformation [57] to analyze the frequencies contained in the sampled time-series data of scan traffic volume, which is collected by the control center mentioned in Section VI.2.c. If there is no worm propagation on the network, the background traffic volume, as white noise, should have equal (expected) strengths on all frequency components (i.e., from low to high frequency). If a worm is propagating, however, there will be a strong low-frequency component in the frequency domain, because of the continuous and exponential growth of worm-generated traffic volume (which can be considered as having a very large period). Thus, the spectrum-based scheme detects worm propagation by identifying low-frequency components with high power spectrum.

Formally, let $r(t)$ be the traffic volume collected at time t. At time t_0, the control center has collected a time-series data set $\{r(0), r(1), \ldots, r(t_0)\}$. We transform the time-series data to the frequency domain using the *discrete Fourier* transform [57] as follows: for all integer

$$s(k) = \sum_{n=0}^{t_0} r(n) \cdot e^{-\frac{2\pi i}{t_0+1} kn}, \quad \text{(VI-18)}$$

where *s(k)* are the transformed frequency component corresponding to period $2\pi k/(t_0 + 1)$, and *i* is the imaginary unit. If $r(t)$ is consisted of white noise only, the expected complex modulus of $s(k)$ (i.e., $|s(k)|$) should be the same for all k in $[0, t_0]$. Nonetheless, when a worm is propagating, the expected $|s(k)|$ for lower frequencies (i.e., large k) will be larger than higher frequencies. Thus, in order to detect worm propagation, we need to measure the differences between $|s(k)|$ for difference frequency ranges.

In particular, we use a widely adopted measure in pattern recognition called Spectral Flatness Measure (SFM) [49], which is defined as the ratio between the geometric mean and the arithmetic mean of s(k).

$$SFM = \frac{[\prod_{k=0}^{t_0} s(k)]^{\frac{1}{t_0+1}}}{\frac{1}{t_0+1}\sum_{k=0}^{t_0} s(k)}. \quad \text{(VI-19)}$$

Generally speaking, the smaller SFM is, the more difference there is between $s(k)$ at different frequency ranges [49], and thus the more likely it is that a worm is propagating on the network. As such, our spectrum-based detection scheme issues an alert when the value of SFM is smaller than or equal to a pre-determined threshold T_M. Note that the greater T_M is, the more false alarms will be generated by the spectrum-based approach. Thus, the defender must specify the value of T_M (along with T_R for the threshold-based scheme) based on the maximum tolerable false alarm rate δ.

Since the value of SFM decreases when the worm propagator adopts a higher growth rate for a longer period of time, we assume, for the sake of simplicity, that at time t_0, $SFM \leq T_M$ if and only if the worm uses $p(t) > p_M$ for a (cumulated) period longer than $\gamma_M \cdot t_0$ time slots (p_M, γ_M in [0,

1]). The values of p_M and γ_M depend on the defender-specified threshold T_M. The larger T_M is, the smaller p_M and γ_M will be.

Note that this spectrum-based scheme can be easily integrated with the threshold-based and trace-back schemes in the framework proposed in Section VI.2. In particular, the control center will perform both the threshold-based and spectrum-based schemes based on collected data, and issues an alert if either scheme generates an alarm. After detecting a propagating worm, it issues an order to initiate the trace-back process.

6.c. Threshold-Based, Trace-Back, and Spectrum-Based Schemes

We now show that an integration of the threshold-based, trace-back, and spectrum-based schemes can effectively defend against the propagation of dynamic self-adaptive worms. In particular, we prove that if the trace-back interval t_B is longer than a (reasonable) threshold, the game will reach Nash equilibrium in the case where the worm propagator will be forced not to propagate any (static or dynamic) self-adaptive worm. Note that with the introduction of the spectrum-based scheme, the strategy set of the defender includes the determination of not only the volume threshold T_R but also the SFM threshold T_M. The strategy set of the worm propagator remains the same. As we mentioned in Section VI.2, the false positive rate Λ now depends on both T_R and T_M.

Let T_M^0 be the maximum threshold for the false positive rate to satisfy $\Lambda \le \delta$ when $T_R = \infty$. Let p_M^0 and γ_M^0 be the corresponding values of p_M and γ_M when $T_M = T_M^0$. Suppose that $f_M^0(t)$ is the number of infected computers at time t when no defender exists in the system, and the worm propagator uses

$$p(t) = \begin{cases} 1, & with\ probabilit\ y\ r_M^0; \\ p_M^0, & with\ probabilit\ y\ 1 - r_M^0. \end{cases} \qquad \text{(VI-20)}$$

for all t in $[0, t_E]$. We have the following theorem.

Theorem VI-4. When the worm propagator propagates a dynamic self-adaptive worm in the system and the defender uses an integration of the threshold-based, trace-back, and spectrum-based schemes, the Nash equilibrium of the game is as follows:

− When $f_M^0(t_E - t_B) \le m$, the worm propagator chooses not to propagate the worm (i.e., $p(t) \equiv 0$). The defender chooses $T_R = \infty$ and $T_M = T_M^0$.

− Otherwise, the worm propagator chooses

$$p(t) = \begin{cases} \min(1, T_R / f(t)), & with\ probabilit\ y\ r_M^0; \\ \min(p_M, T_R / f(t)), & with\ probabilit\ y\ 1 - r_M^0. \end{cases} \quad \text{(VI-21)}$$

The defender chooses the integration of T_R and T_M that i) minimizes $f(t_D)$ when the worm uses the above strategy, and ii) satisfies $\Lambda \le \delta$.

Proof*:* We first consider the case where $f_M^0(t_E - t_B) \le m$. Apparently, the defender already reaches the maximum possible $u_D = 0$ and cannot benefit by changing its strategy. For the worm propagator, suppose that it changes the propagation growth rate function to $p_1(t)$. Let the changed function of the number of infected computers be $f_1(t)$. Due to the definition of spectrum-based scheme and $f_M^0(t)$, there must be $f_1(t) \le f_M^0(t)$ for all t in $[0, t_E]$. Thus,

$$f_1(t_E - t_B) \le f_M^0(t_E - t_B) \le m. \quad \text{(VI-22)}$$

That is, the worm propagator will be traced back with probability of at least 50%, resulting in $u_A = -\infty$. As such, the worm propagator cannot benefit by changing its strategy unilaterally.

We now consider the case where $f_M^0(t_E - t_B) > m$. Note that in order to avoid being detected by the threshold-based scheme, the worm propagator must maintain $p(t) \le T_R / f(t)$. Based on our previous discussion, it is easy to verify that the worm propagator cannot benefit by changing its strategy unilaterally. For the defender, if it changes either T_R or T_M, there will be only two possible outcomes: i) an increased $f(t_D)$, and/or ii) $\Lambda > \delta$. Either way, the defender will have a

decreased utility function u_D. Thus, the defender cannot benefit by changing its strategy unilaterally.

Due to the theorem, with the integration of all three schemes, there are two possible outcomes of worm propagation:

Outcome 1. When t_B is greater than the derived threshold (i.e., satisfies $f_M^0(t_E - t_B) \leq m$), the trace-back and spectrum-based schemes will force the worm propagator not to propagate the worm.

Outcome 2. When t_B does not satisfy the condition, the trace-back scheme poses no threat to the worm propagator. In this case, it is the threshold-based and spectrum-based schemes that force the worm propagator to reduce $p(t)$ to a reasonable level as specified in the theorem.

Table VI-1 Performance of Defensive Strategies

	S_1	S_1+S_2	S_1+S_2+S_3
Traditional worm	Effective	Effective	Effective
Static self-adaptive worm		Effective	Effective
Dynamic self-adaptive worm			Effective

S_1: Threshold-based scheme; S_2: Trace-back scheme; S_3: Spectrum-based scheme

We now analyze which outcome is likely to occur in practice based on practical examples. In particular, we demonstrate that the derived threshold on the trace-back interval t_B in *Outcome 1* is reasonable in many practical systems: We use the same system setting as the one used in Sections 6.a and 6.b. Based on the simulation results, there is T_M^0= 72,000, p_M^0= 0.22 and γ_M^0= 0.5. Due to the theorem, the worm propagator will not propagate the worm as long as $t_B \geq 1.8$

days. As we mentioned in Section 2, this trace-back interval is reasonable in practice. Thus, the integration of all three schemes can effectively defend against dynamic self-adaptive worms in the system, as shown in Table VI-1.

7. Performance Evaluation

In this section, we present the simulation results of systems with static and dynamic self-adaptive worms. In particular, we conduct the simulation on a combination of real-world background scan traffic and simulated worm generated traffic.

For the background scan traffic, we use the real-world DShield logs dataset provided by the SANs Internet storm center (ISC) [25]. The dataset contains more than 80 million scan records, with a size of over 80 GB. All scan records are captured between January 1, 2005 and January 15, 2005. Each record includes the source IP address, destination IP address, destination port number, and time stamp of a monitored scan.

With the real-world scan traces serving as the background traffic, we add simulated worm generated traffic as follows: We use the same system setting as the one specified in Section 5: The number of vulnerable computers on the Internet is 350,000. The total number of IP addresses is 4.3×10^9. The scan rate of worm propagation is 358 scans/minute. The maximum false positive rate is 2%. The maximum propagation time is t_E = 5 days. We conduct the simulation based on various trace-back parameters, with $m = 0.002 \cdot N$ or $0.005 \cdot N$ and the maximum trace-back interval t_E ranging from 1,400 to 7,000 minutes.

We measure the performance of our countermeasures by the maximum infection rate when the worm propagator chooses the optimal strategy of propagation growth rate as specified in the Nash equilibrium. Recall that the maximum infection rate is defined as the ratio of the number of

infected computers to the total number of vulnerable computers at the moment when the worm is detected, or at time t_E, whichever comes first.

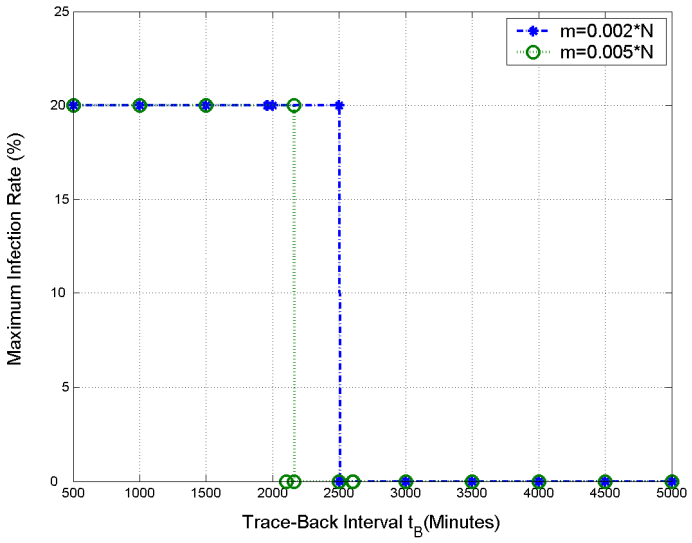


Fig. VI-1. Maximum Infection Rate for Static Self-Adaptive Worm

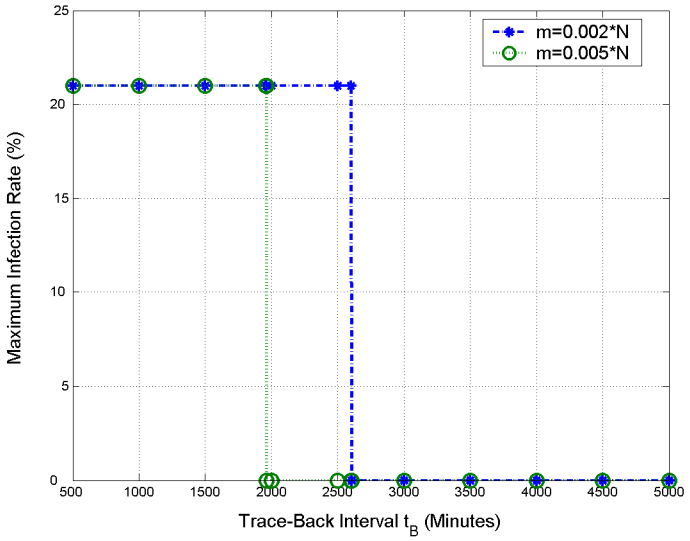


Fig. VI-2. Maximum Infection Rate for Dynamic Self-Adaptive Worms

We present the simulation results of our countermeasures on static self-adaptive and dynamic self-adaptive worms, respectively. For static self-adaptive worms, we measure the performance of an integration of the threshold-based and trace-back schemes. We also compare the results with previous approaches that use threshold-based scheme only [31]. The simulation results are shown in Fig. VI-1. As we can see from this figure, when the trace-back interval t_E is longer than 1.45 days when $m = 0.005 \cdot N$ or 1.81 days when $m = 0.002 \cdot N$, the worm propagator will be forced to not propagate the worm. As we discussed in Section 5, such trace-back interval is reasonable in practice. Thus, an integration of the threshold-based and trace-back schemes can defend against static self-adaptive worms effectively. On the other hand, if only threshold-based scheme is available, the number of infected computers is more than 71,400 (20.4% of all vulnerable computers). As we can see, the threshold-based scheme cannot defend itself against static self-adaptive worms effectively.

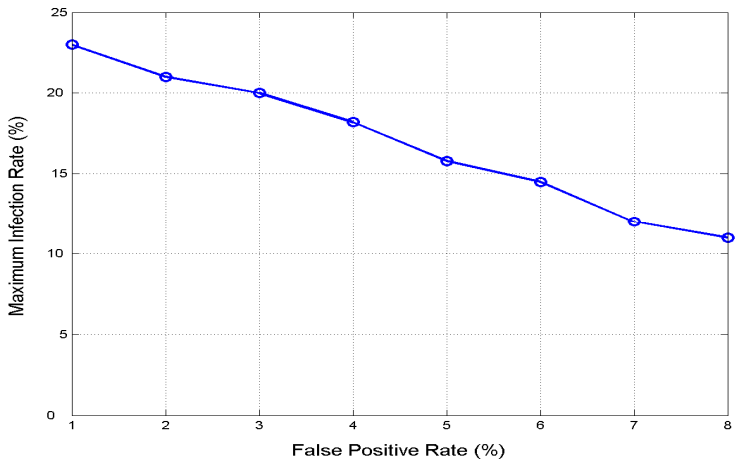


Fig. VI-3. Relationship Between Maximum Infection Rate and Maximum False Positive Rate

For static self-adaptive worms, we measure the performance of an integration of all three, threshold-based, traceback, and spectrum-based, schemes. The simulation results are shown in Fig. VI-2. As we can see from this figure, when the trace-back interval t_B is longer than 1.36 days when $m = 0.005 \cdot N$ or 1.81 days when $m = 0.002 \cdot N$, the worm propagator will be forced to not propagate the worm. As we discussed in Section 5, such trace-back interval is reasonable in practice. Thus, an integration of all three schemes can effectively defend against dynamic self-adaptive worms.

In Fig. VI-3, we also investigate the relationship between the maximum infection rate and the maximum tolerable false positive rate δ when the trace-back interval is not enough to eliminate worm propagation. As we can see from the figure, the more false alarms the system can tolerate, the less that computers can be infected by dynamic self-adaptive worms. In particular, the maximum tolerable false positive rate increases from 1% to 8% and the maximum information rate decreases from 23% to 11% of all vulnerable computers.

8. Extensions

We now discuss how to generalize the utility function of the worm propagator which we proposed in Section 3. Note that in Section 3, we assumed that the worm propagator either receives infinite penalty from trace back (i.e., $u_A = -\infty$ when $f(t_D - t_B) > m$), or none at all (when $f(t_D - t_B) > m$). In practice, however, different worm propagators may differently evaluate the risk of being traced back. Some risk-averse worm propagators may stop propagating the worm when the probability of being traced back is 10%, while others may choose to propagate regardless of whether or not they will be traced back. Thus, we generalize the utility function of a worm propagator to a continuous function, in order to model the threats from worm propagators with different risk aversion levels.

In particular, let $h(x)$ be the loss of the worm propagator if the defender can trace-back to x infected computers at the earliest trace-back time $\max(0, t_D - t_B)$. Apparently, $h(x)$ should be monotonically decreasing with x, as a larger x makes it more difficult to identify the worm propagator. Let $\alpha > 0$ be a preferential parameter pre-determined by the worm propagator. The generalized objective of a worm propagator is to maximize

$$U_A = f(t_D) - \alpha \cdot h(f \max(0, t_D - t_B))). \tag{VI-23}$$

As we can see, our utility function defined in Section 3 is a special case of this generalized version when h is defined as follows:

$$h(x) = \begin{cases} \infty, & if \quad x \le m; \\ 0, & otherwise. \end{cases} \tag{VI-24}$$

Given the generalized utility function, Theorem VI-3 and Theorem VI-1 can be restated as follows:

Theorem VI-5. When the worm propagator propagates a static self-adaptive worm in the system and the defender uses an integration of the threshold-based and trace-back schemes, the Nash equilibrium of the game is as follows:

− When

$$\alpha \cdot h(\frac{N(e^{\beta \cdot p_E \cdot Nt_E})^{1-q}}{(e^{\beta \cdot p_E \cdot Nt_E})^{1-q} + N} > p_E T_R^0, \tag{VI-25}$$

the worm propagator chooses not to propagate the worm (i.e., $p = 0$). The defender chooses $T_R = T_R^0$.

− Otherwise, the worm propagator chooses $p = p_E$. The defender chooses $T_R = T_R^0$.

Theorem VI-6. When the worm propagator propagates a dynamic self-adaptive worm in the system and the defender uses an integration of the threshold-based, trace-back, and spectrum-based schemes, the Nash equilibrium of the game is as follows:

− If there exists T_M and T_R such that 1) $\alpha \cdot h(f(t-t_B)) > p_E T_R$, and 2) the false positive rate $\Lambda < \delta$, then the worm propagator chooses not to propagate the worm (i.e., $p(t) \equiv 0$). The defender chooses the corresponding T_R and T_M.

− Otherwise, the worm propagator chooses

$$p(t) = \begin{cases} \min(1, T_R / f(t)), & with\ probabilit\ y\ r_M; \\ \min(p_M, T_R / f(t)), & with\ probabilit\ y\ 1 - r_M. \end{cases} \quad \text{(VI-26)}$$

The defender chooses the integration of T_R and T_M that 1) minimizes $f(t_D)$ when the worm uses the above strategy, and 2) satisfies $\Lambda \leq \delta$.

The basic idea of proving the above two theorems is similar to the proof of Theorem VI-2 and Theorem VI-3. The optimal strategy for the worm propagator is to select the maximum propagation growth rate p or $p(t)$ that delays the detection time to t_E. The condition for a static self-adaptive worm to stop the propagation is to make utility function, defined in (VI-23) less than 0.

9. Summary

In this chapter, we studied the countermeasure based on contradicted objectives of worm attacks. In particular, we considered a general form of worms called self-adaptive worms, which adapt their propagation patterns to avoid detection. Based on the degree of control on the propagation growth rate, we classified self-adaptive worms into two general categories: static self-adaptive worms and dynamic ones. We demonstrated that existing worm detection schemes are insufficient to counteract self-adaptive worms. Based on a game-theoretic formulation of the

interaction between the worm propagator and the defender, we showed that an effective integration of multiple defensive schemes is critical for defending against self-adaptive worms, which can force the worm attacker to choose the contradicted objectives. To this end, we considered three schemes: threshold-based scheme, trace-back scheme, and spectrum-based scheme. We showed that the combination of the first two schemes can be used to defend against static self-adaptive worms, while the combination of all three schemes can effectively defend against dynamic self-adaptive worms.

CHAPTER VII

COUNTERMEASURE BASED ON THE DEFENDER'S REPUTATION

In this chapter, we focus on developing the countermeasure based on the defender's reputation to defend against worm attacks.

1. Overview

The real-world worm defense systems usually face constant threats from multiple emerging worm attackers. The war between the worm attacker and defender can be treated as a never-ending process with iterative interactions between the two sides. One side tries to adapt itself in order to defeat the other. Studies in previous chapters show that an intelligent attacker can evolve itself and degrade the performance of detection systems. For example, in Chapter VI, we show that worm attackers may adaptively manipulate their propagation traffic pattern or payload to avoid detection and to infect more computers.

In this chapter, we consider real-world system settings with multiple incoming worm attackers that collaborate by sharing the history of their interactions with the defender. We propose a novel countermeasure approach to actually improve the performance of detection system over time by establishing the defender's reputation of toughness in its repeated interactions with multiple incoming worm attackers. Our studies show that while such iterative attacks may enable an attacker to learn from the previous interactions, the defender can also take advantage of the iteration by sacrificing short-term performance in the initial few rounds to establish a "tough" reputation, in return for much higher payoff in the long-run by using the established reputation to force subsequent worm attackers to drop their attacks.

We first formalize the problem as a repeated game between one long-term player (defender) and multiple short-term players (attackers). With the model of repeated games, we define the

defender's reputation as the attackers' estimation of the toughness of the defender. Then, we classify the repeated games into two categories based on whether the attackers have complete information about the defender's objectives. For each category, we propose a generic reputation-aware scheme to optimize the long-term performance of worm defense systems by establishing the defender's reputation in the initial rounds of interactions. Our reputation-aware schemes are transparent to the underlying detection algorithms, and thus can also be used with various other network security applications.

In the following, we first present our system models, introduce a game-theoretic formulation of the repeated interactions between the defender and the worm attackers, as well as the concept of a defender's reputation of toughness, and classify the repeated games into two cases based on the completeness of information in the games. We then propose two reputation-aware worm detection schemes for these two types of games, respectively, and present theoretical analysis of their performances, followed by numerical evaluation of our proposed schemes and conclusion. Notice that this Chapter is based on the joined work between Texas A&M University and the University of Texas at Arlington. My work focused on the problem definition, algorithms design, worm detection evaluation, and literature survey.

2. Models

In this section, we introduce our system models. We first define the participating parties, and then present the strategies and objectives of the parties.

2.a. Parties

Let there be one defender D and n worm attackers $A_1, \ldots, A_n$ in the system. For the sake of simplicity, we assume that each attacker launches no more than one attack to the system. By

using worm detection mechanisms, the defender may detect an attack but may also generate false alarms, thus damaging system functionalities.

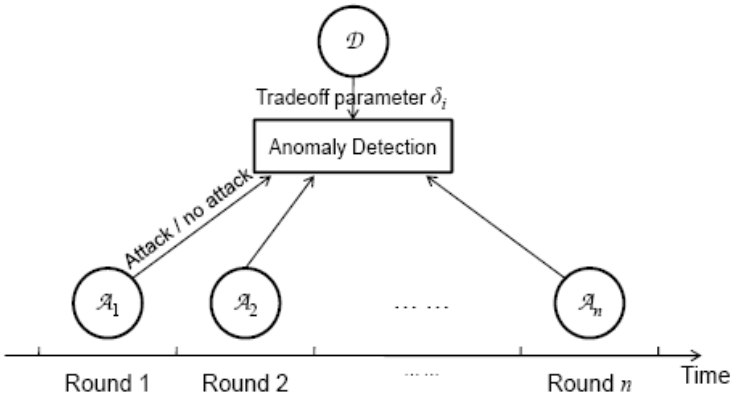


Fig. VII-1. Multiple Round System Architecture

As in real-world systems where each attacker may launch its attack at a different time, we consider the attacks to be iteratively carried out in a group of independent processes. Without loss of generality, we assume that these processes are executed in a serializable manner. Thus, we consider n rounds of interactions, each of which takes place between the defender and one attacker. In particular, we assume that attacker A_i (i in [1, n]) interacts with (i.e., either launches an attack or chooses not to attack) the defender at Round i. If A_i launches an attack, the attack is either detected or missed by anomaly detection by the beginning of Round $i + 1$.

In practice, different attackers have their own interests but may share information, such as the outcomes of previous attacks. As such, we assume the attackers to be independent but cooperative. They are independent in the sense that each attacker aims to maximize its own payoff (see the objective functions in Section 3.c. for details). They are cooperative in the sense that all attackers share their information about the system, including the results of all previous

attacks. Again, we will briefly discuss in Section 5.c the extension of our results to cases where certain attackers are fully cooperative in that they work as a single entity to maximize their joint benefits.

Fig. VII-1 describes the basic architecture of the system, where n attackers sequentially interact with the defender. The definition of tradeoff parameter δ_i for the defender will be introduced in the next subsection.

2.b. Strategies

The strategy of each attacker A_i is to determine whether to launch an attack at Round i. In particular, such a decision may be made based upon observations on the interactions between the defender and the preceding attackers (i.e., A_1 through A_{i-1}). The strategy of defender D is to determine a proper tradeoff between the detection rate and the false alarm rate. We assume that the defender uses a tradeoff parameter δ_i in [0, 1] to control such tradeoff in Round i. The higher δ_i is, the less false alarms are issued. Nonetheless, the defender also has smaller probability to detect an attack launched by A_i. Without loss of generality, we assume that the probability for an attack to be detected at Round i is $1 - \delta_i$ (otherwise we can always normalize δ_i to satisfy this assumption). As such, when $\delta_i = 0$, all targeted attacks will be detected while the maximum tolerable amount of false alarms will be issued. When $\delta_i = 1$, no false alarm will be issued while no attack will be detected. The defender D may determine δ_i based on observations on preceding interactions in Rounds 1 to $i - 1$.

The tradeoff parameter δ_i models a wide variety of tradeoff control mechanisms in real-world applications. For example, δ_i can be considered as threshold on a feature (e.g., traffic volume or other properties) modeled in normal system profile and monitored by the defender, such that the defender issues an alert whenever the observed feature exceeds the threshold. This is a primary

method to control the tradeoff in research and practices [110]. δ_i can also be considered as an output of distributed anomaly detection algorithms, such as the probability of anomaly predicted by Bayesian detection [111].

The combination of strategies for the defender and an attacker determines the outcome of their interaction, which may be one of the following possibilities: i) attack launched and detected, ii) attack launched and not detected, iii) attack not launched. Such outcomes are observed and recorded by both the defender and all the attackers. Note that the outcome does not indicate whether a false alarm is triggered. The reason is that as in most practical systems, we assume that the attackers cannot observe the activation of false alarms.

It is noteworthy that the strategies of preceding attackers (i.e., attack/no attack) can be inferred from the observed outcomes, and are, therefore, public. Nonetheless, the strategy of the defender (i.e., the value of δ_i) is not directly observed by the attackers, especially when an attack is not launched in Round *i*. As such, the attackers can only infer the defender's strategy based on the outcomes of preceding attacks.

2.c. Objectives

The objective of each attacker A_i is to launch an undetected attack at Round *i*. Formally, the objective of A_i is to maximize its utility function $u_A(i)$, which is defined as follows:

$$u_A(i) = \begin{cases} 0, & if\ attack\ not\ launched; \\ 1, & launched\ and\ undetected; \\ -\beta_A, & launched\ and\ detected. \end{cases} \quad \text{(VII-1)}$$

where β_A is a predetermined preference parameter for the attacker. We assume that $\beta_A > 0$ because, otherwise, an attacker will always choose to launch its attack. We believe that this assumption resembles the scenarios of many real-world applications where an attacker may prefer not launching an attack that will always be detected (which may lead to punishment of the

attacker, as demonstrated by recent events [103, 104]). When different attackers have different values of β_A, we assume β_A to be the minimum possible value.

The defender *D* has two objectives: i) to detect all attacks, and ii) to prevent false alarms from being issued. Formally, let

$$l_A(i) = \begin{cases} 1, & \textit{if attack at Round } i \textit{ is un}\det\textit{ected}; \\ 0, & \textit{otherwise.} \end{cases} \quad \text{(VII-2)}$$

Note that due to our definition of δ_i, $l_A(i) = \delta_i$ if an attack is launched at Round *i*.

Since the number of false alarms only depends on the value of δ_i, let $l_F(\delta_i)$ in [0, 1] be a monotonically decreasing function that measures the number of false alarms at Round *i*. The greater $l_F(\delta_i)$ is, the more (or more probability of) false alarms are generated in Round *i*. Without loss of generality, we assume that the number of false alarms generated at Round *i* reaches the maximum tolerable threshold when $l_F(\delta_i) = 1$. When $l_F(\delta_i) = 0$, no false alarm is generated at Round *i*.

Formally, the objective of defender *D* is to maximize its utility function u_D defined over the *n* rounds as follows:

$$u_D = \sum_{i=1}^{n} u_D(i) = -\sum_{i=1}^{n} \beta_D l_F(\delta_i) + (1-\beta_D) l_A(i), \quad \text{(VII-3)}$$

where $u_D(i)$ is the payoff of the defender at Round *i*, and β_D in [0, 1] is the preference parameter for the defender which measures its preference between detection rate and false alarm rate. The greater β_D is, the more concerns the defender has on false alarm rate. In particular, a defender with $\beta_D = 1$ does not care about the detection of attacks while a defender with $\beta_D = 0$ does not care about the loss from false alarms.

3. Reputation in Game-Theoretic Formulation

In this section, we will introduce the concept of reputation in repeated interactions. In particular, we will first present a game-theoretic framework which formulates the repeated interactions between a defender and multiple incoming attackers. With the model of repeated games, we define the defender's reputation based on the attackers' estimation of the preference parameter of the defender. Then, we classify the repeated games into two categories based on whether the attackers have complete information about the defender's objectives.

3.a. Game-Theoretic Formulation

As we mentioned in Section 2, the defender faces attacks from n incoming attackers in an iterative fashion. Thus, we formulate the system as a non-cooperative n-round repeated game between one long-time player, the defender, and n short-term players, the attackers. The game is non-cooperative [108] because there are no coalitions or contracts between the defender and the attackers enforced through outside parties. Each round of the game follows the Stackelberg leadership model [108] with the defender being the leader and the attacker being the follower. This is because, in real-world anomaly detection systems, the defender always moves first by determining its detection tradeoff parameter δ_i before an attacker launches the attack. Note that the defender knows *ex ante* that the follower observes the existence of anomaly detection. The objectives of the players and the set of their possible strategies are defined in Section 2.

Based on the game-theoretic formulation, we have the following theorem on the Nash equilibrium of the game when there is only one round of interaction (i.e., $n = 1$) and the defender knows β_D as pre-knowledge. Remark that Nash equilibrium represents states where neither party can benefit by deviating from the protocol unilaterally.

Theorem VII-1. When there is only one round of interaction and the attacker knows β_D as pre-knowledge, the Nash equilibrium is formed by an attacking strategy that launches an attack if and only if there exists δ in (β_A /(1 + β_A), 1], such that

$$\beta_D > \frac{\delta}{2\delta + l_F(\frac{\beta_A}{1+\beta_A}) - l_F(\delta)}. \tag{VII-4}$$

and a defensive strategy sets

$$\delta_0(\beta_D) = \begin{cases} \arg\max(u_D^A(\delta)), & if\ (VII-4)\ holds; \\ \dfrac{\beta_A}{1+\beta_A}, & otherwise; \end{cases} \tag{VII-5}$$

where $u_A^D(\delta)$ is the defender's payoff if an attack is launched:

$$u_D^A(\delta) = -\beta_D \cdot l_F(\delta) - (1-\beta_D)\cdot\delta. \tag{VII-6}$$

Proof: Recall that as we mentioned in the game-theoretic formulation, each round follows the Stackelberg leadership model where the attacker is the follower that responds to the leader's (i.e., defender's) strategy. Thus, we first prove that for the given defensive strategy in the theorem, the attacking strategy is optimal. Note that when (VII-4) holds, there must be

$$\delta_0(\beta_D) = \arg\max_\delta u_D^A(\delta) \geq \frac{\beta_A}{1+\beta_A}, \tag{VII-7}$$

due to the monotonically increasing property of $l_F(.)$. As such, the expected utility of the attacker A_i is $u_A(i) = \delta_0(\beta_D) - (1 - \delta_0(\beta_D))\cdot\beta_A \geq 0$. Since the attacker's utility by not launching an attack is 0, the specified strategy of launching an attack is optimal. Similarly, we can prove that when (VII-4) does not hold, there is $u_A(i) \leq 0$ when A_i launches its attack. Thus, for the given defensive strategy, the attacking strategy specified in the theorem is optimal.

We now prove that the defensive strategy in the theorem is optimal. We consider two cases respectively: When (VI-4) does not hold, the utility of the defender at the round is:

$$u_D = -\beta_D l_F(\frac{\beta_A}{1+\beta_A}). \quad \text{(VII-8)}$$

If the defender can benefit by changing the tradeoff parameter to δ', there must be $\delta' > \delta'_0(\beta_D) = \beta_A/(1+\beta_A)$ because $l_F(.)$ is monotonically increasing. Nonetheless, since the attacker is the follower, it will then respond by launching the attack, as its expected payoff from an attack will become greater than 0. Note that when the attacker launches its attack, the maximum possible payoff for the defender is $\max_\delta u_D^A(\delta)$, which is smaller than u_D in (VII-8) when (VII-4) does not hold. Thus, the defender cannot benefit by deviating from $\delta_0(\beta_D)$.

When (VII-4) holds, the defender cannot benefit by changing its tradeoff parameter if the attacker launches its attack. In order to force the attacker not to launch its attack, the defender must choose $\delta' \le \beta_A/(1+\beta_A)$. Nonetheless, doing so will not benefit the defender because

$$-\beta_D l_F(\delta') \le -\beta_D l_F(\frac{\beta_A}{1+\beta_A}) \le u_D^A(\delta_0(\beta_D)). \quad \text{(VII-9)}$$

Thus, the defensive strategy specified in the theorem is also optimal.

The defensive strategy $\delta_0(\beta_D)$ in the theorem represents a local optimal strategy when the payoff of only one round is considered. It also represents the optimal defensive strategy if the defender does not evolve its strategy over time in repeated interactions.

From this theorem, we have following observations. When $\beta_D = 1$, the defender will always choose $\delta_i = 1$ to minimize false alarm rate and make $u_D(1) = 0$. When $\beta_D = 0$, however, the defender will choose $\delta_i = 0$ to detect all attacks. In turn, when the attackers know the value of β_D, they will choose to launch every attack when $\beta_D = 1$, but not to launch any attack when $\beta_D = 0$, because the expected gain from an launched attack is always less than 0 when $\delta_i = 0$. As we can see from (VII-4), the strategy of the attacker depends on the knowledge (or estimation) of β_D.

This motivates us to propose a scheme where the defender manipulates the attackers' estimation of β_D in order to control their attack strategies.

3.b. Reputation

From an attacker's perspective, a defender with lower (or greater) β_D is "tougher" (or "softer"). We speculate that while an attacker may launch an attack to a soft defender, it may choose not to do so when the defender is tougher. Thus, we define the reputation of a defender as an attacker's estimation on the defender's preference parameter β_D. Formally, we have the following definition.

Definition 1. The reputation of the defender at the beginning of Round i, $r_D(i)$, is defined as the posterior expected value of β_D based on the outcomes of Rounds 1 to $i - 1$:

$$r_D(i) = \int_0^1 x p(x \mid outcomes\ of\ Rounds\ 1,...,i-1)dx. \quad \text{(VII-10)}$$

where $p(.)$ is the posterior probability density function of β_D based on the outcomes of Rounds 1 to $i - 1$.

We now prove the above speculation by showing the influence of the defender's reputation on the attackers' strategies. In particular, we consider a simple defensive strategy to choose between $\delta_i = 0$ or 1 based on β_D (this simplified setting will be important for the analysis of our proposed schemes). In this case, we have the following theorem.

Theorem VII-2. When the defender chooses δ_i in {0, 1}, an attacker A_i will not launch attack if

$$r_D(i) \le \frac{\beta_A}{\beta_A + 1}. \quad \text{(VII-11)}$$

Proof: We will prove that when (VI-11) holds, the attacking strategy of not launching attack and a defensive strategy of setting

$$\delta_i = \begin{cases} 0, & \textit{with probability of } 1-\beta_D; \\ 1, & \textit{with probability of } \beta_D; \end{cases} \quad \text{(VII-12)}$$

forms Nash equilibrium of the game. It is noteworthy that since the defender can observe the outcome of every previous interaction, the value of $r_D(i)$ is known to the defender.

Since each round follows the Stackelberg leadership model with the attacker being the follower, similar to the proof of Theorem VII-1, we first prove that for the given defensive strategy, the attacking strategy of not launching attack is optimal. With the defensive strategy, from the perspective of attacker A_i, the expected probability of its attack being detected is $1 - r_D(i)$. Thus, when A_i launches the attack, its expected utility function is

$$Exp(u_A(i)) = 1 \cdot r_D(i) - \beta_A \cdot (1 - r_D(i)). \quad \text{(VII-13)}$$

When an adversary A_i chooses not to launch its attack, the expected utility of A_i is 0. As we can see, $\text{Exp}(u_A(i)) \leq 0$ if and only if $r_D(i) \leq \beta_A/(\beta_A + 1)$. Thus, when (VI-11) holds, the attacker cannot receive any benefit by unilaterally changing its strategy to launch its attack. Thus, for the given defensive strategy, the attacking strategy of not launching attack is optimal.

We now prove that the defender cannot benefit by unilaterally changing its strategy either. When the defender chooses the strategy in (VI-12), its utility function is

$$u_D(i) = -\beta_D \cdot (1 - \beta_D). \quad \text{(VII-14)}$$

Suppose that a defender can increase its utility function by changing δ_i to δ'. Note that since the attacker will not launch its attack when $\delta_i = \beta_D$, there must be $\delta' > \beta_D$. Nonetheless, the attacker (as the follower) will respond by choosing to launch its attack because its expected payoff will be greater than 0. In this case, the utility function of the defender satisfies

$$u'_D(i) = -\beta_D \cdot (1 - \delta) - (1 - \beta_D) \cdot \delta' < -\beta_D \cdot (1 - \beta_D) = u_D(i). \quad \text{(VII-15)}$$

As such, the defender cannot benefit by unilaterally changing its strategy either. Thus, the attacking strategy of not launching attack and the defensive strategy in (VI-12) form Nash equilibrium of the game. That is, no attacker will launch the attack when (VI-11) holds.

The Theorem VII-2 confirms our speculation that a tougher reputation (i.e., smaller $r_D(i)$) may prevent certain attackers from launching attacks. Thus, the basic idea of our reputation-aware anomaly detection schemes presented in the next two sections is to reduce $r_D(i)$ by manipulating defensive strategies in the initial rounds of interactions, in return for much higher payoff in the long-run.

3.c. Classification of Games

Since we aim to reduce $r_D(i)$ which is the attackers' estimation of β_D, the attacker's pre-knowledge about β_D is critical to the effectiveness of reputation-aware anomaly detection. Thus, before introducing reputation-aware schemes, we first classify the repeated games into two categories based on the attackers' pre-knowledge about β_D:

- Case A: In this case, the attackers do not know the exact value of β_D (before Round 1), and can only estimate the value based on i) a prior distribution of β_D, and ii) observed interactions. Since the attackers do not know the utility function of the defender (which depends on β_D) as pre-knowledge, the games between the defender and the attackers contain incomplete information.
- Case B: In this case, the attackers know the exact value of β_D as pre-knowledge. As such, the games between the defender and the attackers contain complete information.

In the following two sections, we will introduce reputation-aware anomaly detection schemes for the above two cases.

4. Reputation-Aware Worm Detection: Case A

In this section, we will introduce our reputation-aware worm detection scheme for Case A, where the defender's preference parameter β_D is unknown to the attackers. We will first present the detection algorithm, and then analyze its performance theoretically. Numerical evaluation of the algorithm will be presented in Section 6.

4.a. Algorithm A

In Case A, the attackers have no pre-knowledge about β_D. An attacker can only estimate $r_D(i)$ based on the outcomes of previous interactions as well as the prior distribution of β_D. Thus, our basic idea is for a soft defender to simulate the behavior of a tougher one in the initial rounds of interactions, in order to reduce $r_D(i)$ and to build a tough reputation.

Algorithm VII.A: for Case A

1: STATUS ← UNESTABLISHED.

2: for each Round i do

3: if STATUS = ESTABLISHED then

4: $\delta_i \leftarrow 1$ if $\beta_D = \beta_0$; $\delta_i \leftarrow 0$ if $\beta_D = 0$.

5: else if STATUS = EXPIRED then

6: $\delta_i \leftarrow \delta_0(\beta_D)$.

7: else if $\beta_D = \beta_0$ then

8: $\delta_i \leftarrow 0$ with probability of $p\beta_A/(p\beta_B - p + 1)$ otherwise.

9: else if $\beta_D = 0$ then

10: $\delta_i \leftarrow 1$ with probability of $p\beta_A/(p\beta_B - p + 1)$, 0 otherwise.

11: end if

12: Set δ_i as the tradeoff parameter for Round i.

13: Wait until an attack succeeds or is detected.

14: if STATUS = UNESTABLISHED and R(i) > i/2 then

15: STATUS ← ESTABLISHED.

16: else if STATUS = UNESTABLISHED and $i \geq n_0$ then

17: STATUS ← EXPIRED.

18: end if

19: end for

For the sake of simplicity, we assume β_D to be either 0 (i.e., extremely tough) with probability of p or $\beta_0 > 0$ (i.e., relatively soft) otherwise. Since an extremely tough defender with $\beta_D = 0$ always chooses $\delta_i = 0$, we only need to consider the cases where $p < 1/(1+\beta_A)$ because otherwise no attacker will launch attack due to Theorem VII-2.

Algorithm VII.A depicts our reputation-aware anomaly detection scheme for Case A. In the algorithm, $R(i)$ is the number of detected attacks in Rounds 1 to i, n_0 is a pre-determined parameter on the number of rounds the defender intends to use to build its reputation, and $\delta_0(\beta_D)$ is the local optimum derived in Theorem VII-1. To help better understand the algorithm, we call a defender tough if $\beta_D = 0$ and as soft if $\beta_D = \beta_0$.

At the initial rounds (when STATUS = UNESTABLISHED), a soft defender chooses $\delta_i = 0$ with probability of $p\beta_A/(p\beta_A - p + 1)$ while a tough one does so with probability of $1 - p/(p\,\beta_A - p + 1)$. Once more than half of the previously launched attacks are detected (i.e., $R(i) > i/2$), the reputation of toughness is considered to be established (i.e., STATUS = ESTABLISHED). Then, a tough defender always chooses $\delta_i = 0$ while a soft one chooses $\delta_i = 1$. Note that once STATUS becomes ESTABLISHED, it is never set to other values. If the reputation is not established by

the end of Round n_0 (i.e., STATUS = EXPIRED), the defender returns to its local optimum $\delta_0(\beta_D)$.

As we can see from Algorithm VII.A, our reputation-aware scheme considers the anomaly detection algorithm as a black box with input of δ_i. Thus, our scheme is transparent to the underlying anomaly detection algorithms and can be used in various anomaly detection applications.

We now briefly explain the reputation-building mechanism in Algorithm VII.A: When STATUS = UNESTABLISHED, the strategy for a soft defender is tougher than its local optimal strategy $\delta_0(\beta_D)$, while the strategy of a tough one is softer than its local optimum. Such deviation (from local optimum) is designed to reduce $r_D(i)$ when the defender is soft and to thereby allow a soft defender to establish a reputation of toughness. As a result, we have the following theorem:

Theorem VII-3. When STATUS = ESTABLISHED at Round i,

$$r_D(i) = \frac{\beta_A}{1+\beta_A}. \tag{VII-16}$$

Proof: Suppose that STATUS = ESTABLISHED at the beginning of Round i while STATUS = UNESTABLISHED at the beginning of Round $i - 1$. Due to Algorithm A, there must be at least $[i/2]$ detected attacks in Rounds 1 to $i - 1$. Suppose that the number of detected attacks is d ($d \geq [i/2]$) and

$$p_R = \frac{p\beta_A}{p\beta_A - p + 1}. \tag{VII-17}$$

We have

$$
\begin{aligned}
r_D(i) &= \frac{\beta_0(1-p)p_R^d(1-p_R)^{i-1-d}}{p(1-p_R)^d P_R^{i-1-d} + (1-p)p_R^d(1-p_R)^{i-1-d}} \\
&\le \frac{\beta_0(1-p)p_R}{p(1-p_R)+(1-p)p_R} \\
&= \frac{\beta_0(1-p)p\beta_A}{p(1-p)+(1-p)p\beta_A} \\
&\le \frac{\beta_0\beta_A}{1+\beta_A} \le \frac{\beta_A}{1+\beta_A}.
\end{aligned}
\tag{VII-18}
$$

Note that due to Theorem VI-2, no attacker will launch its attack when $r_D(i)$ satisfies (VII-16). Without further observable interaction, $r_D(i)$ will remain the same after Round i. Thus, (VII-16) holds whenever STATUS = ESTABLISHED at Round i.

Due to Theorem VII-2 and VII-3, after STATUS = ESTABLISHED, no subsequent attacker will launch attacks to the system.

As we can see, when STATUS = ESTABLISHED, a soft defender will not issue any false alarm and will also not present any undetected attack. Thus, the expected utility of a defender is 0 for all subsequent rounds, higher than the utility of local optimum when $\beta_D = \beta_0$. Thus, by sacrificing the utility when STATUS = UNESTABLISHED in some initial rounds for building reputation, the defender can obtain payback in later rounds due to the established reputation.

4.b. Theoretical Analysis

As we mentioned above, a key property of Algorithm A is that no attacker will launch attack when STATUS = ESTABLISHED. Thus, we first derive the probability for STATUS = ESTABLISHED at the end of Round n_0.

Theorem VII-4. Given $\beta_D = \beta_0$, when n_0 is sufficiently large, the probability that STATUS = ESTABLISHED after Round n_0 is at least $\beta p_A/(1 - p)$.

Proof: Due to Algorithm A, STATUS is either ESTABLISHED or EXPIRED after Round n_0. Let $f(n_0)$ be the probability that STATUS = EXPIRED after Round n_0. Note that STATUS = EXPIRED if and only if there exists $i < n_0$, $R(i) \leq i/2$.

We now derive $f(n_0)$ by transforming the problem to the monotonic path counting problem in combinatorics. Consider a grid with $n \times n$ square cells in Fig. VII-2. We start with the lower left corner at Round 1. If an attack is detected (i.e., $\delta_i = 0$), we move one step right along an edge of the grid. If an attack is not detected (i.e., $\delta_i = 1$), we move one step up. As we can see, if $R(i) \leq i/2$ holds for all $i < n_0$, then the path never crosses the diagonal of the grid. Thus, in order to derive the probability of STATUS = EXPIRED, we need to count the number of paths that satisfy the condition. Without loss of generality, we assume that n0 is even. Note that when n0 is odd, then $f(n_0) = f(n_0 + 1)$. At the end of Round n_0, the finishing point of the path can be $(n_0, 0)$, $(n_0 - 1, 1)$, . . ., $(n_0/2, n_0/2)$. Note that when $x, y \geq 1$ and $x \geq y$, the number of monotonic paths from (0, 0) to (x, y) which never crosses the diagonal is

$$g(x,y) = \binom{x+y}{y} - \binom{x+y}{y-1}. \tag{VII-19}$$

Since the number of monotonic path from (0, 0) to $(n_0, 0)$ is 1, let $g(n_0, 0) = 1$. Suppose that

$$p_R = \frac{p\beta_A}{p\beta_A - p + 1}. \tag{VII-20}$$

The probability that STATUS = EXPIRED after Round n_0 is

$$\begin{aligned} f(n_0) &= \sum_{y=0}^{n_{0/2}} p_R^y (1-p_R)^{n_0-y} g(n_0 - y, y) \\ &= \sum_{y=0}^{n_{0/2}} p_R^y (1-p_R)^{n_0-y} \binom{n_0}{y} - \sum_{y=0}^{n_{0/2}} p_R^y (1-p_R)^{n_0-y} \binom{n_0}{y-1}. \end{aligned} \tag{VII-21}$$

Note that the first component of (VII-21) is the cumulative probability from $y = 0$ to $y = n_0/2$ for a binomial distribution with mean $n_0 p_R$ and variance $n_0 p_R(1-p_R)$. When n_0 is sufficiently

large, such binomial distribution can be approximated by a normal distribution with the same mean and variance. Thus, we have

$$f(n_0) = \frac{1-2p_R}{2(1-p_R)}\left(1+erf\left(\frac{n_0(1-2p)^2}{8p(1-p)}\right)\right) \le \frac{1-2p_R}{1-p_R}. \quad \text{(VII-22)}$$

where erf(.) is the Gaussian error function. That is, the probability for STATUS = ESTABLISHED after Round n_0 is at least $1 - f(n_0) = p\beta_A/(1 - p)$.

As we can see from the theorem, when n_0 is sufficiently large, there is a fairly large probability for STATUS to be ESTABLISHED, such that no subsequent attacker will launch attacks while no false alarms will be issued by a soft defender. For example, when $p = 1/3$ and $\beta_A = 1$, the probability of no launched attack after Round n_0 is at least 1/2 when $n_0 \to \infty$. In fact, as we will show in Section 6, the probability of no launched attack increases quickly with n_0.

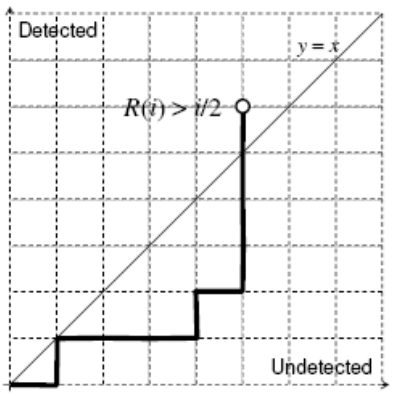


Fig. VII-2. $n \times n$ Grid

Based on the theorem, we have the following corollary on the utility function of the defender.

Corollary VII-1. If n_0 is sufficiently large, when Algorithm A is used, the expected payoff of a soft defender satisfies

$$\lim_{n\to\infty}\frac{u_D}{n}\geq\frac{1-p-p\beta_A}{1-p}\max_{\delta}u_D^A(\delta). \quad \text{(VII-23)}$$

Proof: Due to Theorem VII-2, VII-3, and VII-4, when n_0 is sufficiently large, the probability that STATUS = EXPIRED after Round n_0 is at most $(1-p-p\beta_A)/(1-p)$. Note that when STATUS = ESTABLISHED, the expected payoff of a soft defender is 0 because no attacks will be launched while no false alarm will be triggered (due to $\delta_i = 1$). When STATUS = EXPIRED, the expected payoff of a soft defender is $\max_\delta u_A^D(\delta)$. Thus, the expected payoff of a soft defender satisfies

$$\lim_{n\to\infty}\frac{u_D}{n}\geq\frac{1-p-p\beta_A}{1-p}\max_{\delta}u_D^A(\delta). \quad \text{(VII-28)}$$

when n_0 is sufficiently large.

4.c. Extension

We now briefly discuss the extension of Algorithm VII.A to a wider variety of system settings, where an attacker may launch multiple attacks, and multiple fully cooperative attackers may commit to their joint (rather than individual benefits). Note that these two cases are essentially the same as we can always model attacks from fully cooperative attackers as multiple attacks launched by a single attacker.

For these system settings, the only change required for Algorithm VII.A is to assign the same δ_i for all attacks launched by the same attacker. By doing so, an attacker cannot obtain a better estimation of β_D by launching multiple attacks because the outcomes for all of its subsequent attacks are exactly the same as the outcome of its first attack.

As we can see, as long as each attacker can only launch finite number of attacks, Theorem VII-4 and Corollary VII-1 always hold, with the only exception being that the required n_0 may be larger due to the number of (essentially) duplicate attackers launched by an attacker.

5. Reputation-Aware Worm Detection: Case B

In this section, we will introduce our reputation-aware worm detection scheme for Case B, where the preference parameter β_D is known by the attackers as pre-knowledge. We will first present the detection algorithm, and then analyze its performance theoretically. Numerical evaluation of the algorithm will be presented in Section 6.

5.a. Algorithm B

Algorithm VII.B: for Case B

1: if β_D does not satisfy (VII-4) then

2: use the local optimal strategy in Theorem VII-1 and exit;

3: else

4: Randomly choose β_D based on (VII-29).

5: STATUS ← UNESTABLISHED.

6: end if

7: for each Round i do

8: if STATUS = ESTABLISHED then

9: $\delta_i \leftarrow 0$ if $\beta_R = 0$, $\delta_i \leftarrow 1$ if $\beta_R = 1$.

10: else if STATUS = EXPIRED then

11: $\delta_i \leftarrow \delta_0(\beta_D)$.

12: else if $\beta_D = 1$ then

13: $\delta_i \leftarrow 0$ with probability of p_R, $\delta_i \leftarrow 1$ otherwise.

14: else if $\beta_D = 0$ then

15: $\delta_i \leftarrow 1$ with probability of p_R, $\delta_i \leftarrow 0$ otherwise.

16: end if

17: Set δ_i as the tradeoff parameter for Round i.

18: Wait until an attack succeeds or is detected.

19: if STATUS = UNESTABLISHED and R(i) > i/2 then

20: STATUS ← ESTABLISHED.

21: else if STATUS = UNESTABLISHED and i ≥ n_0 then

22: STATUS ← EXPIRED.

23: end if

24: end for

In Case B, the attackers knows the exact value of β_D as preknowledge. Due to Theorem VII-1, an attacker will only attack a defender with preference parameter satisfying (VII-4). Thus, we only need to consider these defenders in this section.

Algorithm VII.B depicts our reputation-aware anomaly detection scheme for Case B. For the sake of simplicity, we assume that a defender may only choose between $\delta_i = 0$ and 1, but may mix the two choices with certain probability distribution. In the algorithm, $R(i)$, n_0, and $\delta_0(\beta_D)$ have the same meaning as in Algorithm VII.A, β_R is chosen randomly based on the following distribution:

$$\beta_R = \begin{cases} 0, & \text{with probability } 1-\sqrt{\beta_D}; \\ 1, & \text{with probability } \sqrt{\beta_D}; \end{cases} \tag{VII-29}$$

and

$$p_R = \frac{\beta_A(1-\sqrt{\beta_D})}{(\beta_A-1)(1-\sqrt{\beta_D})+1}. \tag{VII-30}$$

In order for the defender to establish a reputation of toughness, in Algorithm B, we first introduce uncertainty to the defender's toughness by a random parameter β_R. According to the algorithm, unless STATUS = EXPIRED, the defender chooses its strategy based on β_R instead of its real preference parameter β_D. This requires the attackers to estimate β_R in order to respond to the defensive strategy, and opens spaces for the defender to establish its reputation.

Specifically, to help better understand the algorithm, we refer a defender as "tough" if the random parameter $\beta_R = 0$ and as soft if $\beta_R = 1$. Note that Algorithm B is essentially similar to Algorithm A with β_R replacing β_D. At the initial rounds (when STATUS = UNESTABLISHED), a soft defender chooses $\delta_i = 0$ with probability of p_R while a tough one does so with probability of $1 - p_R$. Once more than half of the previously launched attacks are detected (i.e., $R(i) > i/2$), the reputation is considered to be established (i.e., STATUS = ESTABLISHED). Then, a tough defender always chooses $\delta_i = 0$ while a soft one chooses $\delta_i = 1$. If the reputation is not established by the end of Round n0 (STATUS = EXPIRED), the defender's strategy returns to its local optimum $\delta_0(\beta_D)$.

Suppose that $r_R(i)$ is defined in analogy to $r_D(i)$ as the attackers' estimation of β_R. Similar to Algorithm A, the basic idea of Algorithm B is to establish reputation of toughness (i.e., reduce β_R) by deviating from the local optimal strategy. We have the following theorem for Algorithm B:

Theorem IV-5. When STATUS = ESTABLISHED at Round i,

$$r_R(i) = \frac{\beta_A}{1+\beta_A}. \tag{VII-31}$$

Proof: Suppose that STATUS = ESTABLISHED at the beginning of Round i while STATUS = UNESTABLISHED at the beginning of Round $i - 1$. Due to Algorithm B, there

must be at least [i/2] detected attacks in Rounds 1 to i − 1. Suppose that the number of detected attacks is d ($d \geq$ [i/2]) and

$$p_R(i) = \frac{p\beta_A}{p\beta_A - p + 1}. \tag{VII-32}$$

We have

$$\begin{aligned} r_D(i) &= \frac{\sqrt{\beta_D}\, p_R^d (1-p_R)^{i-1-d}}{(1-\sqrt{\beta_D})(1-p_R)^d\, p_R^{i-1-d} + \sqrt{\beta_D}\, p_R^d (1-p_R)^{i-1-d}} \\ &\leq \frac{\sqrt{\beta_D}\, p_R}{(1-\sqrt{\beta_D})(1-p_R) + \sqrt{\beta_D}\, p_R} \\ &= \frac{\sqrt{\beta_D}\beta_A(1-\sqrt{\beta_D})}{(1-\sqrt{\beta_D})\sqrt{\beta_D} + \sqrt{\beta_D}\beta_A(1-\sqrt{\beta_D})} \\ &\leq \frac{\beta_A}{1+\beta_A}. \end{aligned} \tag{VII-33}$$

Note that due to Theorem VII-3, no attacker will launch its attack when $r_D(i)$ satisfies (VII-16). Without further observable interaction, $r_D(i)$ will remain the same after Round i. Thus, (VII-16) holds whenever STATUS = ESTABLISHED at Round i. Due to Theorem VII-2 and VII.5, after STATUS = ESTABLISHED, no subsequent attacker will launch attack to the system.

Note that a soft defender obtains payback once the reputation is established. As we can see, if STATUS = ESTABLISHED at Round i, the expected utility of a defender is

$$u_D(i) = -\beta_D \cdot \Pr(\beta_R = 0) = -\beta_D(1-\sqrt{\beta_D}). \tag{VII-34}$$

For a defender of concern in Case B (i.e., satisfies (VII-4)), this is always greater than the expected utility (β_D − 1) from the one-round local optimum $\delta_0(\beta_D)$.

5.b. Theoretical Analysis

Similar to the analysis of Algorithm A, we first derive the probability for STATUS = ESTABLISHED at Round n_0:

Theorem VII-6. When n_0 is sufficiently large, the probability that STATUS = ESTABLISHED after Round n_0 is at least

$$\beta_A(1-\sqrt{\beta_D})/\sqrt{\beta_D}. \tag{VII-35}$$

Proof: In analogy the proof of Theorem VII-4, we can prove that the probability that STATUS = EXPIRED after Round n_0 is

$$f(n_0) \le \frac{1-2p_R}{1-p_R}. \tag{VII-36}$$

Note that for Algorithm B,

$$p_R = \frac{\beta_A(1-\sqrt{\beta_D})}{(\beta_A-1)(1-\sqrt{\beta_D})+1}. \tag{VII-37}$$

Thus, the probability that STATUS = ESTABLISHED after Round n_0 satisfies

$$1-f(n_0) \ge \frac{p_R}{1-p_R} = \frac{\beta_A(1-\sqrt{\beta_D})}{\sqrt{\beta_D}}. \tag{VII-38}$$

As we can see from the theorem, there is a fairly large probability for STATUS to be ESTABLISHED, which prevents the forthcoming attacker from launching attacks. For example, when $\beta_A = 1$ and $\beta_D = 2/3$, the probability of no launched attack after Round n_0 is at least 22.4%. Based on the theorem, we have the following corollary.

Corollary VII-2. If n_0 is sufficiently large and β_D satisfies (VII-4), when Algorithm B is used, the expected payoff of the defender satisfies

$$\lim_{n\to\infty}\frac{u_D}{n} \ge u_0 + \frac{\beta_A(1-\sqrt{\beta_D})}{\sqrt{\beta_D}}(1-2\beta_D+\beta_D^{3/2}), \tag{VII-39}$$

where u_0 is the utility function of a defender taking local optimal strategy with $\delta_i = \delta_0(\beta_D)$.

Proof: When STATUS = ESTABLISHED after Round n_0, the expected utility of the defender is

$$Exp(u_D(i)) = -(1-\sqrt{\beta_D})\beta_D. \quad \text{(VII-40)}$$

When STATUS = EXPIRED, the expected utility of the defender is

$$u_0 = \beta_D - 1. \quad \text{(VII-41)}$$

Since the probability that STATUS = ESTABLISHED after Round n_0 is at least

$$\beta_A(1-\sqrt{\beta_D})/\sqrt{\beta_D}, \quad \text{(VII-42)}$$

when n_0 is sufficiently large, the expected payoff of the defender satisfies

$$\lim_{n\to\infty}\frac{u_D}{n} \geq u_0 + \frac{\beta_A(1-\sqrt{\beta_D})}{\sqrt{\beta_D}}(1-2\beta_D+\beta_D^{3/2}), \quad \text{(VII-43)}$$

Similar to the extension in Section 4.c, we can also extend Algorithm VII.B to the system settings with attackers launching multiple attacks or fully cooperative attackers. Theorem VII-6 and Corollary VII-2 still hold for these scenarios, with the only exception being that a larger n_0 may be required due to the duplicate attacks launched by an attacker.

6. Performance Evaluation

In this section, we show the derived optimal strategies for the defender and the attackers in the game. The numerical results actually demonstrate the detection rate and false positive rate in a state consisting of the optimal strategies, and thus can be used to demonstrate the real performance of systems using our reputation-aware schemes.

In particular, we compute the numerical results of Algorithms VII.A and VII.B based on a real-world case study of applying our reputation-aware scheme to an existing worm detection approach [31] which detects anomaly of scan traffic generated in worm propagation by issuing an alert when the rate of observed scan traffic exceeds a threshold computed from the

background traffic. Note that with a lower threshold, a worm is more likely to be detected, but a higher false positive rate will also be generated, leading to a tradeoff between detection rate and false positive rate. Again, we would like to remark that in this chapter, we are not promoting any specific anomaly detection algorithm. Instead, we use the case study to show that the incorporation of a defender's reputation can enable defensive schemes that achieve better tradeoff between detection rate and false positive rate.

Table VII-1. Tradeoff between Detection Rate and Fase Positive Rate

Threshold Ratio (r)	1.2	1.5	1.8	2.1	2.4	2.7	3
Detection Rate	0.86	0.72	0.58	0.44	0.29	0.15	0.01
False Positive Rate	0.98	0.93	0.67	0.38	0.27	0.14	0.1

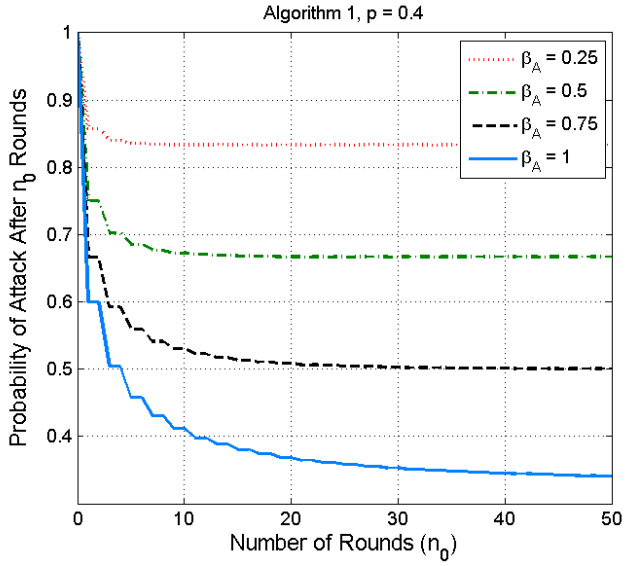


Fig. VII-3. Probability of Attacker Launching Attack After Round n_0

In order to determine the numerical values of $l_F(.)$, we use a real-world log of traffic data, the DShield logs provided by SANs ISC as background traffic [25]. It includes records of scan recorded between January 1, 2005 and January 15, 2005. We use data on port 80 as an example. According to the background traffic recorded by the DShield traffic logs, the mean and variance of the number of scan packets recorded per minute is m = 31 and σ^2 = 92.97, respectively. We consider a pure-random-scan worm targeting a population of 350,000 vulnerable hosts on the Internet with 100 scans per minute. We define the detection rate as the probability that a worm is detected within 600 minutes after the start of its propagation. With detection threshold (on observed scan traffic) being $m+r\cdot\sigma$, where r in [1, 4] is the threshold ratio, we compute the tradeoff between detection rate and false alarm rate, some examples of which are shown in Table VII-1. Note that the values are normalized to [0, 1].

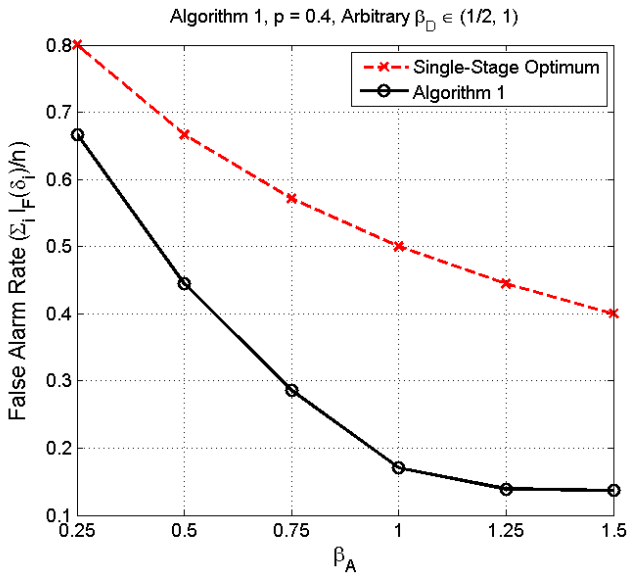


Fig. VII-4. Comparison between Algorithm A and Local Optimal Strategy δ_0

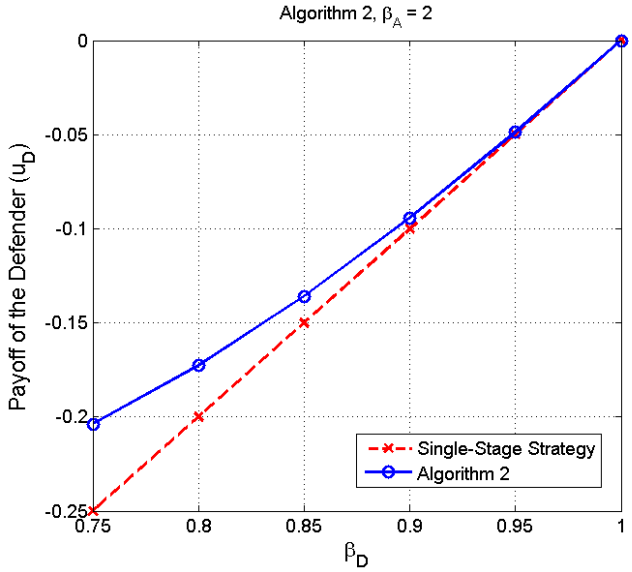


Fig. VII-5. Comparison between Algorithm B and Local Optimal Strategy δ_0

Based on the data, Fig. VII-3 shows the probability that an attacker A_i with $i > n_0$ launches its attack when Algorithm A is used. We demonstrated the cases where n_0 ranges from 1 to 50, the defender's β_D satisfies Pr{ $\beta_D = 0$} = p = 0.2, and the attacker's preference parameter β_A in {1, 2, 3, 4}. As we can see, the probability of attack decreases rapidly while n_0 increases. In particular, when β_A = 4, the probability that an attacker launches an attack after 50 rounds is less than 11.3%.

We also evaluate the performance of Algorithm A based on the false alarm rate required to force all attackers after Round n_0 not to launch their attacks. Fig. VII-4 shows the false alarm rates for Algorithm VII.A and the local (one-round) optimal strategy δ_0 when n_0 = 50, p = 0.2, and the attacker's preference parameter β_A ranges from 1 to 9. As we can see, our reputation-aware scheme in Algorithm A significantly reduces the number of generated false alarms. In particular, when β_A = 9, the false alarm rate of Algorithm A is only 18.3% of the local optimal defensive strategy δ_0.

For systems where the defender's preference parameter is already known by the attackers, when Algorithm VII.B is used, the loss of the defender (from missed attacks and false alarms) is shown in Fig. VII-5. We set β_A = 2 and n_0 = 50. We compare the loss with the local optimal defensive strategy δ_0. As shown in the figure, our reputation-aware scheme reduces the loss of defender, especially when β_D is small. When β_D is large, the defender has no concern about detection rate, making the reputation of toughness less useful. Thus, the performance of Algorithm B converges to that of the local optimum when $\beta_D \rightarrow 1$.

7. Summary

In this chapter, we proposed the countermeasure based on establishing the defender's reputation of toughness to improve the performance of worm detection. We considered real-world system settings with multiple incoming worm attackers that collaborate by sharing the history of their interactions. We formalized such systems through a game-theoretic formulation for the repeated interactions between the defender and multiple worm attackers. Based on the formulation, we proposed generic algorithms to improve the performance of worm detection system by incorporating the defender's reputation. We further classified the repeated games into two categories based on whether the attackers have complete information about the defender's objectives. We presented the basic ideas, detailed algorithms, and theoretical analysis of reputation-aware anomaly detection approaches for the two categories. We demonstrated the effectiveness of our scheme by numerical studies on the study of worm detection. Our data validates our findings and indicate that incorporating reputation can significantly improve the performance of anomaly detection systems. As part of our future work, we are applying this framework to investigate the defender's reputation and game theory analysis on other security applications and systems.

CHAPTER VIII

CONCLUDING REMARKS

In this dissertation, we have systematically studied countermeasures against worm attacks, namely traffic-based and non-traffic based countermeasures. For traffic-based countermeasures, we propose our approaches and develop countermeasures by identifying some key features of worm propagation and probing attack traffic. For non-traffic based countermeasures, we propose approaches that robustly capture dynamic signatures of worm program execution, test a feature of contradicted objectives, and incorporate a defender's ability to defend against worm attacks.

This dissertation develops a framework that allows us to study both traffic related features and non-traffic related features and, hence, to develop countermeasures against worm attacks. The problems addressed in the proposed research are important, both theoretically and practically. Particularly, the developed results lay the theoretical foundation for countermeasures of worm attacks and help us to understand problem and solution space. The techniques developed for countermeasures are practical and hence can be applied to real-world systems.

REFERENCES

[1] D. Moore, C. Shannon, and J. Brown, "Code-red: a case study on the spread and victims of an internet worm," in *Proc. of the 2nd Internet Measurement Workshop (IMW)*, Marseille, France, Nov. 2002.

[2] D. Moore, V. Paxson, and S. Savage, "Inside the slammer worm," *IEEE Magazine of Security and Privacy*, vol. 1, no. 4, pp. 33-39, 2003.

[3] CERT, CERT/CC advisories, [Online]. Available: http://www.cert.org/advisories/. Accessed on March 2005.

[4] W32/MyDoom.B Virus, [Online]. Available: http://www.us-cert.gov/cas/techalerts/TA04-028A.html. Accessed on March 2004.

[5] The Honeynet Project and Research Alliance, Know your enemy: Tracking botnets, Online]. Available: http://www.honeynet.org/papers/bots/. Accessed on January 2005.

[6] P. R. Roberts, Zotob Arrest Breaks Credit Card Fraud Ring, [Online]. Available: http://www.eweek.com/article2/0,1895,1854162,00.asp. Accessed on March 2004.

[7] R. Naraine, Botnet Hunters Search for Command and Control Servers, [Online]. Available: http://www.eweek.com/article2/0,1759,1829347,00.asp. Accessed on March 2004.

[8] W32.Sircam.Worm@mm, [Online]. Available: http://www.symantec.com/avcenter/venc/data/w32.sircam.worm@mm.html. Accessed on March 2004.

[9] Worm.ExploreZip, [Online]. Available: http://www.symantec.com/avcenter/venc/data/worm.explore.zip.html. Accessed on March 2004.

[10] Z. S. Chen, L.X. Gao, and K. Kwiat, "Modeling the spread of active worms," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, CA, Mar. 2003.

[11] C. C. Zou, W. Gong, and D. Towsley, "Worm propagation modeling and analysis under dynamic quarantine defense," in *Proc. of the 1st ACM CCS Workshop on Rapid Malcode (WORM)*, Washington DC, Oct. 2003.

[12] S. Staniford, "Containment of scanning worms in enterprise networks," *Journal of Computer Security*, vol. 3, no. 5, pp. 321-355, Mar. 2001.

[13] J. Twucrpss and M. M. Williamson, "Implementing and testing a virus throttling," in *Proc. of the 12th USENIX Security Symposium (SECURITY)*, Washington, DC, Aug. 2003.

[14] S. G. Chen and Y. Tang, "Slowing down internet worms," in Proc. of the *24th International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, Mar. 2004.

[15] Trend Micro, [Online]. [Online] Available: http://www.trendmicro.com. Accessed on March 2004.

[16] J. O. Kephard and S. R. White, "Directed-graph epidemiological models of computer virus," in *Proc. of 1991 Computer Society Symposium on Research in Security and Privacy (S&P)*, Oakland, CA, May 1991.

[17] J. O. Kephart and S. R. White, "Measuring and modeling computer virus prevalence," in Proc. of the *14th IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 1993.

[18] S. Staniford, V. Paxson, and N. Weaver, "How to own the Internet in your spare time," in *Proc. of the 11th USENIX Security Symposium (SECURITY)*, San Francisco, CA, Aug. 2002.

[19] C. C. Zou, W. Gong, and D. Towsley, "Code-red worm propagation modeling and analysis," in *Proc. of the 9th ACM Conference on Computer and Communication Security (CCS)*, Alexandria, VA, Nov. 2002.

[20] R. Perdisci, O. Kolesnikov, P. Fogla, M. Sharif, and W. Lee, "Polymorphic blending attacks," in *Proc. of the 15th USENIX Security Symposium (SECURITY)*, Vancouver, B.C., Aug. 2006.

[21] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and utility of internet sinks for network abuse monitoring," in *Proc. of Symposium on Recent Advances in Intrusion Detection (RAID)*, Pittsburgh, PA, Sept. 2003.

[22] D. Moore, "Network telescopes: Observing small or distant security events," in *Invited Presentation at the 11th USENIX Security Symposium (SECURITY)*, San Francisco, CA, Aug. 2002.

[23] CAIDA, Dynamic Graphs of the Nimda Worm, [Online]. Available: http://www.caida.org/dynamic/analysis/security/nimda. Accessed on May 2004.

[24] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet motion sensor: A distributed blackhole monitoring system," in *Proc. of the 12th IEEE Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, Feb. 2005.

[25] SANS, Internet Storm Center, [Online]. Available: http://isc.sans.org/. Accessed on January 2006.

[26] CAIDA, Telescope Analysis, [Online]. Available: http://www.caida.org/analysis/security/telescope. Accessed on January 2006.

[27] J. Bethencourt, J. Frankin, and M. Vernon, "Mapping internet sensors with probe response attacks," in *Proc. of the 14th USNIX Security Symposium (SECURITY)*, Baltimore, MD, Jul.-Aug. 2005.

[28] Y. Shinoda, K. Ikai, and M. Itoh, "Vulnerabilities of passive internet threat monitors," in *Proc. of the 14th USNIX Security Symposium (SECURITY)*, Baltimore, MD, Jul.-Aug. 2005.

[29] T. Kohno, A. Broido, and K.C. Clafy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp 93-108, Apr.-Jun. 2005.

[30] C. Zou, W. B. Gong, D. Towsley, and L. X. Gao, "Monitoring and early detection for internet worms," in *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS)*, Washington DC, Oct. 2003.

[31] S. Venkataraman, D. Song, P. Gibbons, and A. Blum, "New streaming algorithms for superspreader detection," in *Proc. of the 12th IEEE Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, Feb. 2005.

[32] J. Wu, S. Vangala, and L. X. Gao, "An effective architecture and algorithm for detecting worms with various scan techniques," in *Proc. of the 11th IEEE Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2004.

[33] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distribution," in *Proc. of ACM SIGCOMM*, Philadelphia, PA, Aug. 2005.

[34] S. S Kim and A. L. N. Reddy, "Image-based anomaly detection technique: Algorithm, implementation and effectiveness," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1942-1954, Oct. 2006.

[35] S. S. Kim, A. L. N. Reddy, and M. Vannucci, "Detecting traffic anomalies through aggregate analysis of packet header data", in *Proceedings of Networking'04, Lecture Notes in Computer Science (LNCS)*, Athens, Greece, May 2004.

[36] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. of the ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, Dec. 2004.

[37] H. Kim and B. Karp, "Autograph: toward automated, distributed worm signature detection," in *Proc. of the 13th USENIX Security Symposium (SECURITY)*, San Diego, CA, Aug. 2004.

[38] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Seattle, WA, Aug. 2004.

[39] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2001.

[40] H. H Feng, J. T. Giffin, Y. Huang, S. Jha, W. Lee, and B. P. Miller, "Formalizing sensitivity in static analysis for intrusion detection," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2004.

[41] D. Wagner and D. Dean, "Intrusion detection via static analysis," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2001.

[42] D. Gao, M. Reiter, and Dawn Song, "Behavioral distance for intrusion detection," in *Proc. of Symposium on Recent Advance in Intrusion Detection (RAID)*, Seattle, WA, Sep. 1999.

[43] G. M. Voelker J. Ma and S. Savage, "Self-stopping worms," in *Proc. of the ACM Workshop on Rapid Malcode (WORM)*, Washington D.C, Nov. 2005.

[44] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proc. of USENIX Workshop on Hot Topics in Understanding Botnets (HotBots)*, Cambridge, MA, Apr. 2007.

[45] P. Wang, S. SParka, and C. Zou, "An advanced hybrid peer-to-peer botnet," in *Proc. of USENIX Workshop on Hot Topics in Understanding Botnets (HotBots)*, Cambridge, MA, Apr. 2007.

[46] D. J. Daley and J. Gani, *Epidemic Modeling: An Introduction*, Cambridge University Press, 1999.

[47] S. Soundararajan and D. L. Wang, "A schema-based model for phonemic restoration," *Tech. Report, OSU-CISRC-1/04-TR03*, Department of Computer Science and Engineering, The Ohio State University, Jan. 2004.

[48] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice Hall, 1984.

[49] R. E. Yantorno, K. R. Krishnamachari, J. M. Lovekin, D. S. Benincasa, and S. J. Wenndt, "The spectral autocorrelation peak valley ratio (sapvr) - a usable speech measure employed as a co-channel detection system," in *Proc. of IEEE International Workshop on Intelligent Signal Processing (WISP)*, Budapest, Hungary, May 2001.

[50] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Second Edition, Elsevier Science, 2003.

[51] L. Y. Chuang, C. H. Yang, C. H. Yang, and S. L Lin, "An interactive training system for morse code users," in *Proc. of Internet and Multimedia Systems and Applications*, Honolulu, Hawai, Aug. 2002.

[52] Dshield, Distributed Intrusion Detection System, [Online]. Available: http://www.dshield.org/. Accessed on January 2006.

[53] R. K. Pickholtz, D. L. Schilling, and L. B. Milstein, "Theory of spead-spectrum communication - tutorial," *IEEE Transaction on Communication*, vol. 30, no.5, pp. 8550-8584, Mar. 1982.

[54] E. J. Crusellers, M. Soriano, and J. L. Melus, "Spreading codes generator for wireless cdma network," *International Journal of Wireless Personal Communications*, vol. 7, no. 1, pp 135-142, Jan. 1998.

[55] R. Dixon, *Spread Spectrum Systems*, 2nd Edition, John Wiley & Sons, 1984.

[56] Nova Engineering, Linear Feedback Register Shift, [Online]. Available: http://www.sss-mag.com/pdf /lfsr.pdf. Accessed on January 2005.

[57] R. L. Allen and D. W. Mills, Signal Analysis: Time, Frequency, Scale, and Structure, Wiley and Sons, 2004.

[58] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949.

[59] X. Y. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proc. of the 2007 IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2007.

[60] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading Worm Signature Generators Using Deliberate Noise Injection", In *Proc. of the 2006 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006

[61] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control flow graph matching," in *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, Berlin, Germany, Jul. 2006.

[62] MetaPHOR, [Online]. Available: http://securityresponse.symantec.com/avcenter/venc/data/w32.simile.html. Accessed on January 2006.

[63] P. Ferrie and P. SzÄor. Zmist, Zmist opportunities, Virus Bullettin, [Online]. Available: http://www.virusbtn.com. Accessed on January 2007.

[64] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proc. of the 12th USENIX Security Symposium (SECURITY)*, Washington, DC, Aug. 2003.

[65] M. Christodorescu and S. Jha, "Testing malware detectors," in Proc. of *the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, Boston, MA, Jul. 2004.

[66] M. Ciubotariu, Netsky: a conflict starter?, Virus Bullettin, [Online]. Available: http://www.virusbtn.com, 2004.

[67] J. Gordon, "Lessons from virus developers: The beagle worm history through april 24," [Online]. Available: http://www.securityfocus.com/guest/24228. Accessed on January 2004.

[68] M. Ernst, “Static and dynamic analysis: Synergy and duality,” in *Proc. of ICSE Workshop on Dynamic Analysis*, Portland, OR, May 2003.

[69] S. Li, A Survey on Tools for Binary Code Analysis, Department of Computer Science, Stony Brook University, [Online]. Available: http://www.cs.sunysb.edu/ lshengyi/papers/rpe/RPE.htm, 2004.

[70] H. H. Feng, O. M. Kolesnikov, P.Fogla, W. Lee, and W. Gong, “Anomaly detection using call stack information,” in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2003.

[71] F. Qin, C. Wang, Z. Li, H. Kim, Y. Zhou, and Y. Wu, “Lift: A low-overhead practical information flow tracking system for detecting security attacks,” in *Proc. of IEEE/ACM Annual Symposium on Micro-Architecture (MICRO)*, Orlando, FL, Dec. 2006.

[72] M. H. Dunham, *Data Mining: Introductory and Advanced Topics*, Prentice Hall, First edition, 2002.

[73] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, Second edition, 2006.

[74] W. Lee, S. Stolfo, and Phil Chan, “Learning patterns from unix process execution traces for intrusion detection,” in *Proc. of AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, Menlo Park, CA, Jun. 2003.

[75] W. Lee, S. J. Stolfo, and W. Mok, “A data mining framework for building intrusion detection models,” in *Proc. of the 1999 IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 1999.

[76] S. Martin, A. Sewani, B. Nelson, K. Chen, and A. Joseph, “Analyzing behavioral features for email classification,” in *Proc. of the 2rd International Conference on Email and Anti-Span (CEAS)*, Mountain View, CA, Aug. 2003.

[77] S. Yang, J. P. Song, H. Rajamani, T. W. Cho, Y. Zhang, and R. Mooney, “Fast and effective worm fingerprinting via machine learning,” in *Proc. of the 3rd IEEE International Conference on Autonomic Computing (ICAC)*, Dublin, Ireland, Jun. 2006.

[78] K. Julisch and M. Dacier, “Mining intrusion detection alarms for actionable knowledge,” in *Proc. of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Edmonton, Alberta, Jul. 2002.

[79] C. Nachenberg, “Computer virus-antivirus coevolution”, *Communication of the ACM*, vol. 40, pp 31-40, Jan. 1997.

[80] P. Szor and P. Ferrie, “Hunting for metamorphic," in *Proc. of Virus Bulletin Conference*, Mountain View, CA, Sep. 2001.

[81] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk, Polymorphic shellcode engine using spectrum analysis, [Online]. Available: http://www.phrack.org/. Accessed on March 2003.

[82] Ktwo, Admmutate v0.8.4: Shellcode mutation engine, [Online]. Available: http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz. Accessed on March 2001.

[83] M. Sedalo, Jempiscodes: Polymorphic shellcode generator, [Online]. Available: http://securitylab.ru/. Accessed on May 2003.

[84] VMWare Inc., [Online]. Available: http://www.vmware.com/virtual-machine. Accessed on March 2007.

[85] Microsoft, Microsoft Virtual PC, [Online]. Available: http://www.microsoft.com/ windows/ virtualpc/default.mspx. Accessed on March 2007.

[86] Metasploit LLC, Windows System Call Table, [Online]. Available: http://www.metasploit.com/users /opcode/syscalls.html. Accessed on March 2007.

[87] Operating System Inside, Linux System Call Table, [Online]. Available: http://osinside.net/syscall/system call table.htm, 2006. Accessed on March 2007.

[88] P. Thurrott, Windows "Longhorn" FAQ, [Online]. Available: http://www.winsupersite.com/faq /longhorn.asp. Accessed on March 2007.

[89] GNU Project, Linux Function and Macro Index, [Online]. Available: http://www.gnu.org/software/libc/manual/htmlnode/Function-Index.html#Function-Index. Accessed on March 2007.

[90] K. F. Lee and S. Mahajan, *Automatic Speech Recognition: The Development of the SPHINX System*, Springer, 1988.

[91] Z. Su, Q. Yang, Y. Lu, and H. Zhang, "Whatnext: A prediction system for web requests using n-gram sequence models," in *Proc. of the 1st International Conf. on Web Information Systems and Engineering Conference*, Hong Kong, P.R. China, Jun. 2000.

[92] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.

[93] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

[94] V. Vapnik, *Statistical Learning Theory*, John Wiley and Sons, New York, 1998.

[95] C. Burges, "A tutorial on support vector machines for pattern recognition," *Journal of Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 121-167, Mar. 1998.

[96] H. Nunez, C. Angulo, and A. Catala, "Rule extraction from support vector machines," in *Proc. of European Symposium on Artificial Neural Networks*, Bruges, Belgium, Aug. 2002.

[97] N. Barakat and J. Diederich, "Eclectic rule-extraction from support vector machines," *Journal of Computational Intelligence*, vol. 1, no. 5, pp. 59-62, Apr. 2005.

[98] G. Fung, S. Sandilya, and R. Rao, "Rule extraction from linear support vector machines," in *Proc. of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Chicago, IL, Aug. 2005.

[99] BindView Corporation, Strace for NT, [Online]. Available: http://www.bindview.com/ Services/RAZOR/Utilities/Windows/stracereadme.cfm. Accessed on March 2007.

[100] B. Christian, Full and Naive Bayes Classifiers, [Online]. Available: http://fuzzy.cs.unimagdeburg.de/ borgelt/doc/bayes/bayes.html. Accessed on March 2007.

[101] T. Joachims, *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, 1998.

[102] Zdnet, Smart worm lies low to evade detection, [Online]. Available: http://news.zdnet.co.uk /internet/security/0,39020375,39160285,00.htm. Accessed on January 2004.

[103] V. Sekar, Y. Xie, D. Maltz, M. Reiter, and H. Zhang, "Toward a framework for internet forensic analysis," in *Proc. of the 3rd Workshop on Hot Topics in Networks (HotNets-III)*, San Diego, CA, Nov. 2004.

[104] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang, "Worm origin identification using random moonwalks," in *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, Oakland, CA, May 2005.

[105] Y. Xie, V. Sekar, M. Reiter, and H. Zhang, "Forensic analysis for epidemic attacks in federated networks," in *Proc. of 14th IEEE International Conference on Network Protocols (ICNP)*, Santa Barbara, CA, November 2006.

[106] Cisco, Worm mitigation technical details, [Online]. Available: http://www.cisco.com/ web/about/security/intelligence/worm-mitigation-whitepaper.html. Accessed on March 2004.

[107] Arbor Networks, The Peakflow Platform, Associated Press for Fox News, [Online]. Available: http://www.arbornetworks.com. Accessed on March 2007.

[108] J. Farlow, J. E. Hall, J. M. McDill, and B. H. West, *Differential Equations and Linear Algebra*, Prentice Hall, Inc., 1999.

[109] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, 1994.

[110] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, Sep. 1999.

[111] S. Axelsson, “The base-rate fallacy and its implications for the difficulty of intrusion detection,” in *Proc. of the 6th ACM Computer and Communications Security Conference (CCS)*, Singapore, Nov. 1999.

VITA

Wei Yu received a B.S. degree in Electrical Engineering from Nanjing University of Technology, China, in 1992, an M.S. degree in Electrical Engineering at Tongji University, China, in 1995. He began his doctoral studies in computer science at Texas A&M University in 1999 and received a Ph.D. degree in 2008. His research interests are in the areas of networking and distributed systems with a primary focus on security and privacy, information assurance, computer forensics, and system reliability. Wei Yu may be reached at 4608 Dalroack Dr., Plano, TX 75024, USA. His email address is weyu@cisco.com.