CERIAS Tech Report 2009-33

Analysis of Port Scanning Attacks

by Yu Zhang

Center for Education and Research

Information Assurance and Security

Purdue University, West Lafayette, IN 47907-2086

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By YU ZHANG

Entitled ANALYSIS OF PORT SCANNING ATTACKS

For the degree of DOCTOR OF PHILOSOPHY

Is approved by the final examining committee:

BHARAT BHARGAVA
Chair

DONGYAN XU

XIANGYU ZHANG

VERNON REGO

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Bharat Bhargava

Approved by: Aditya Mathur / William J. Gorman 12/02/2009
Head of the Graduate Program Date

Graduate School Form 20
(Revised 6/09)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:
ANALYSIS OF PORT SCANNING ATTACKS

For the degree of DOCTOR OF PHILOSOPHY

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

YU ZHANG
Printed Name and Signature of Candidate

12/02/2009
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

ANALYSIS OF PORT SCANNING ATTACKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yu Zhang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2009

Purdue University

West Lafayette, Indiana

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

To my parents

ACKNOWLEDGMENTS

This dissertation would not be possible without the support of my Ph.D. advisor, Bharat Bhargava. His help throughout my Ph.D. study has significant impacts on my understanding of research and computer science in general. His experiences inspired and encouraged me to explore unknown areas and try out different ideas.

I am very grateful to Professor Dongyan Xu for his support and help during my Ph.D. study. His comments and suggestions are very helpful to me. Without his help, I would not learn many experiences in a timely manner and advance research. I also would like to express my thanks to Professor Vernon Rego for serving on my Ph.D. committee. His help and knowledge helped me improve the quality of the dissertation significantly. I appreciate the guidance and help from Professor Xiangyu Zhang for the Ph.D. study. His ideas and research methodology influenced me deeply.

Many thanks to Leszek Lilien and Nwokedi Idika for their help during my Ph.D. study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Zhang, Yu. Ph.D., Purdue University, December 2009. Analysis of Port Scanning Attacks. Major Professor: Bharat K. Bhargava.

In this research, we present theoretical models and practical solutions to model and analyze collaborative attacks, with a focus on port scanning attacks and malware propagation.

We study the malware propagation and present results that help understand the effects of Multi-port scanning, Multi-threading, Infection time, Multiple starting points, and Collaboration (MMIMC) on malware propagation. This research quantitatively measures the effects of MMIMC on infected hosts. Experimental results show that the above issues significantly affect malware propagation and verify our analysis.

We discuss architectures, polices, and allocation schemes for collaborative attackers. We present a fast DHT-based collaborative attack scheme that aims to eliminate duplicate attacks, minimize contention, and significantly increase the attack speed. We propose different collaboration strategies and analyze their advantages and disadvantages. We discuss the static, dynamic, and hybrid target selection and allocation schemes. We present the algorithm details and discuss the stop and revisit policies for collaborative attackers.

Our experimental results suggest that collaborative attacks can significantly outperform individual attackers, and provide insights into many design and implementation issues.

1 INTRODUCTION

1.1 Collaboration Among Attackers

The growth of the Internet has rendered its coordination very complex. Security is a key challenge in Internet since most protocols were designed without consideration of any prevention against miscreants. In addition, many emerging technologies make the Internet even more vulnerable, and attacks against networked systems are becoming more complex and powerful. Individual attackers can collaborate to cause more problems for the intruder-identification and defense mechanisms. In this dissertation, we study collaborative attacks [69], [70], [71].

The current approaches to security in network systems deploy individualized security solutions. For example, antiviral software is used to defend against worms and viruses, intrusion detection tools guard against scanning and Denial-of-Service (DoS) attacks, firewalls aim to protect against unwanted connection attempts, and mail filtering tries to foil spam and phishing attempts. Accordingly, most research done today also focuses on improving these individual tools.

An important piece missing from the current research is understanding of ways in which attackers can collaborate to launch attacks.

1.1.1 Some Collaborative Attacks

Collaborative attacks are those launched by multiple malicious adversaries that synchronize their activities to attack network targets. In collaborative attacks, attackers communicate and collaborative with each other to launch much more powerful attacks. For instance, routing attacks can collaborative with Malware attacks and Distributed Denial-of-Service (DDoS) attacks.

In the real world, the following collaborative attacks occurred or could have occurred:

1. One can bring a large number of attackers to increase the computation power. Attackers have employed this approach in the past. For instance, in 1999, more than 100,000 PCs were used to crack the DES challenge of RSA [21].

2. One can assemble a reasonable number of attackers to influence the decision-making of core machines, these include routing and Sybil [72] attacks.

3. One can employ a variety of technologies to launch a full-scale attack. For instance, the coordinated Botnet zombie nodes can collaborate to launch DoS attacks [5], and the well-orchestrated collaborative attacks on Estonia caused large-scale disruptions [13].

Unlike single and un-collaborative group attacks, collaborative attacks may cause more devastating impacts as it combines efforts of more than one attacker (or processes). Examples of attacks include replication attacks, Sybil attacks [72], spam attacks, phishing attacks, worms and viruses, DNS-related attacks, routing-related attacks, Denial-of-Message (DoM) attacks, and DDoS attacks.

Three basic categories of attacks are as follows:

1. *Independent* attacks, which have no knowledge of other attacks. They can be launched at the same time as other attacks but do not know other attacks.

2. *Collaborative* attacks that are coordinated and can be launched simultaneously or sequentially. From the high-level or functional point of view, we further identify the relationships between the launched collaborative attacks and classify them as: (i) non-overlapping (sequential); (ii) partially overlapping; and (iii) fully overlapping. Attacks may target different parts of a network and aim at depleting resources of the defenders. From the low-level or technical point of view (e.g., techniques employed by attackers), attacks can be categorized into:

(i) attacks that may substitute each other; (ii) attacks that may diminish the effects of each other; (iii) attacks that severely damage each other; (iv) attacks that expose other attacks; (v) attacks that should be launched after each other; and (vi) attacks that may target different areas of a network.

3. *Replicated* attacks, in which adversaries can insert additional replicated hostile nodes into a network after obtaining some secret information from the captured nodes or by infiltration. Nodes replicated in this way are likely to uncover the shared secrets of the uncompromised neighboring nodes. Encrypted communication links can be established between a replicated node and the uncompromised nodes. It should be clear that compromising even a single node might allow an adversary to gain partial or even full control of a network by producing many clones and deploying them in the original network.

1.1.2 Dimensions of Attack Taxonomy

Collaborative attacks can be organized into a comprehensive taxonomy. The taxonomy includes a number of essential dimensions:

1. *Attack type:* As already mentioned, the most relevant forms of attacks are: replication attacks, Sybil attacks, DoM attacks and DoS attacks. *Replication attacks* take place when adversaries are able to insert hostile nodes into the network by obtaining some secret information from the captured nodes or by infiltration. *Sybil attacks* occur when a node forges and uses several identities, and in this way obtains a greater control over the network allowing sniffing, packet dropping and delaying packets. *DoS attacks* occur when an attacker floods a server with requests exhausting the server's resources and thus its availability to respond to requests from other nodes.

2. *Attack timing:* Attackers may take advantage of temporal features of the network by choosing periods of higher susceptibility to perform the attack. Also they could coordinate when each attacks to maximize their effectiveness.

3. *Attack severity and strength:* Damage caused by an attack is an important factor in defining the defensive actions to be taken. For instance, an aggressive attack should be handled with a higher priority than non-aggressive attacks.

4. *Attack extent* An attack may affect the whole network or a part of it. The extent of an attack also affects the priority of the actions taken by defenders against it.

5. *Attacker's familiarity with attack target:* Attacks may be conducted by insiders, quite familiar with attack targets, or outsiders. A more detailed categorization may include an attacker who is: a stranger, an acquaintance, a friend, etc. Inflicting damage is easier for an attacker more familiar with the attack target.

6. *Attacker's role:* Attackers can be, for instance, regular users, administrators, or guests.

7. *Ranking of attackers.* Attackers have usually distinct profiles. Some are more effective than others, and some have typical behavior while others are more difficult to characterize.

8. *Composition and coordination of attack activities:* Attackers can exhibit different abilities, including attack coordination abilities. In coordinated, well-organized attacks, attackers with the highest leadership skills will become commanders. Both leaders and followers must share information. How it is done is an important coordination characteristic to be captured in the model of coordination. The graphs of relationships among attackers used in the model can be tree-based and involve inheritance. Coordination lines can be employed to represent coordination.

9. *Communication between attackers:* Attackers can employ checkpointing and synchronization messages to communicate with each other. Coordination lines can again be employed, this time to represent communication. Finding the frequency and interval of attackers' communication can be very useful. Attackers can also utilize independent checkpointing, taking checkpoints of their own. They can also check later offline using other techniques, for instance, out-of-band communication.

10. *Mutual feedback among attackers* In a dynamic environment, coordinated attackers can benefit from exchange of feedback on their attack activities, including information on the results of their attacks. For example, attackers knowing that some ongoing attacks consume many resources of defenders, can adjust their strategy. In this case, the attackers can:

    (i) increase the power of the ongoing attacks; or

    (ii) employ more sophisticated or more focused strategies; or

    (iii) fine-tune the timing of their attacks. Attackers can also adjust the strength of attacks dynamically. For instance, attackers can launch spasmatic attack lasting for a short time, making attack detection and attacker identification very difficult.

11. *Attack and defense strategies:* The number of attackers affects the performance and power of attacks significantly. However, there are situations in which multiple attackers, not properly coordinated, could interfere with each other. Similarly, multiple defenders could also hamper each other. We plan to identify and describe strategies in which coordinated attacks provide synergistic effects, greater than the sum of individual attack effort.

Note that not all the dimensions are required to describe a collaborative attack. For example, The impact of the attacks can be modeled as, impact = f(severity and strength of attack, extent of attack, communication between attackers, attack and defense strategies).

In summary, collaborative attackers can employ a variety of technologies and different collaboration strategies. In this dissertation, we focus on the collaborative port scanning attacks and defense to collaborative attacks.

1.1.3 Port Scanning Attacks

Our focus in this dissertation is the port scanning attacks.

In port scanning attacks, network communication ports on the target hosts are scrutinized by attackers. In an individual port scan, one attacker scans and finds which ports are available on the target machine. In a coordinated port scan, multiple attackers scan and find which ports are available on a number of target machines.

Issues we need to consider for collaborative port scanning attacks include Maximizing network usage, Minimizing latency, and Program Optimization (e.g., how many threads should be employed), etc. We need to consider general issues for distributed systems as well, including synchronization of attack progress, node crashing/failure, and node starvation, etc.

Why do we focus on the port scanning attacks? Some people argue that port scanning attacks are not real attacks. However, we note that:

1. First, port scanning attack is a fundamental form of network attack. One cannot attack without targets;

2. Second, all attackers need to do reconnaissance before their attacks;

3. Third, as soon as attackers discover vulnerable hosts, the actual infection takes little time to occur;

4. Last, offline target discovery is difficult and time-consuming.

A real-world example [4] further illustrates the importance of studying port scanning attacks. A Dutch teenager has employed port scanning to discover specific jailbroken I-phones that are vulnerable to a known vulnerability related to OpenSSH.

The teenager demanded money payment from those I-phone users who have been discovered by the port scanning. Without launching port scanning attacks, it would be impossible for him to find a large number of victims in short notice and receive international attention.

1.2 Related Work

A. Prior Work on Collaborative Attacks

Many researchers have characterized specific Internet attacks or phenomenon using one or more sources of data. For instance, Ref. [73] has characterized spammer behavior. Ref. [74, 75] focus on specific worm outbreaks and Ref. [22] characterizes DoS attacks in the Internet. Very few works have focused on correlating various attacks. One of them is Ref. [76], in which the authors analyze data, logged by the Dshield project [82] on a large number of intrusion detection systems (IDSs), to find out related, possibly collaborative, attacks. Ref. [2] discusses security of WiMAX networks.

B. Coordinated Attacks of SYN Floods and Slammer Worms

A SYN flood attack is launched by sending more TCP connection requests than a target machine can process. A slammer worm uses random scanning to find and infect susceptible hosts.

Both the SYN flood attack and the slammer worm, even if launched separately, can cause significant damage [74, 77]. If they are launched together in a coordinated way, the resulting consequences will be more devastating: the SYN flood attack will effectively block TCP connections while the Slammer worms will propagate via UDP connections. The coupled attack is not only more powerful but also more difficult to deal with.

C. Sybil Attacks

Douceur [72] discusses Sybil attacks, in which a malicious user obtains multiple fake identities and pretends to be multiple, distinct nodes in the system. In this way,

the malicious nodes can control the decisions of the system, especially if the decision process involves voting or any other type of collaboration.

Trust relationships can be created in social networks to limit the number of nodes a malicious node can create. In such an approach, we need to consider trust, security, and privacy issues together, and in a systematic way, preferably at the policy level. In addition, a deliberate collaboration model is needed.

Generic Sybil attacks can be found in Internet as well. For example, BGP would greatly suffer from the aforementioned attacks. Researchers at UCLA have proposed ways to detect invalid routing announcements in RIP [78] but mere detection cannot solve the problem thoroughly. Responding after detection and defending against such attacks, possibly coordinated, remains a challenge.

D. Modeling Multistep Cyber Attacks for Attack Scenario Recognition

Cheung *et al.* [79] state that many cyber attacks can be decomposed into multiple sub-attacks. The authors develop methods and a language for modeling multistep attack scenarios based on typical isolated alerts about attack steps.

The idea of trust relationship [72, 80, 81] is used to limit the number of clones a malicious node can have and defend against Sybil attacks. However, no collaborative model is discussed in these works. In the RIP protocol [78], detection of invalid routing announcements has been suggested. The response after detection and ways to defend against such attacks remains a challenge. Many approaches are proposed. A stochastic model of collaborative internal and external attacks is used in [63]. Data Routing Information (DRI) table and cross checking [64] can be used to identify multiple cooperating black hole nodes. An on-demand routing protocol for ad hoc wireless networks can provide resilience to Byzantine failures caused by individual or colluding nodes [65]. A signature-based model can be used to detect collaborative attacks [69]. Clustering and merging functions can be used to recognize alerts that correspond to the same occurrence of an attack and create a new combined alert [68]. A collaborative system using Multicast, annotated topology information, and blind

detection techniques can be used to detect DDoS attacks [66]. Hidden Markov models can be used to detect collaborative attacks [67].

E. Collaborative attack modeling and attack graph analysis

Bhargava *et al.* used casual model [60] and Lamport proposed event ordering [61] to identify events for concurrency control and synchronization of clocks in distributed systems. Attack graph was proposed to model the order of events. Lippmann *et.al* [62] analyzed most attack graph papers and concluded problems of attack graph analysis: scalability to large networks, generation of attack details, and computing complexity.

2 THE EFFECTS OF THREADING, INFECTION TIME, AND MULTIPLE-ATTACKER COLLABORATION ON ATTACK PROPAGATION

2.1 Introduction

Malware is software designed to compromise computer systems. Examples include Logic Bombs, Viruses, Worms, and Botnets [5], [33]. Malware can be classified into two categories: self-propagating malware and non-self-propagating malware. Self-propagating malware poses a serious threat due to its ability to propagate through networks to infect a large number of hosts. E.g., worms have infected thousands of computers [8], [9], [18], [24]. Malware replicates itself and intrudes vulnerable hosts without human intervention. Malware can carry malicious payloads that can be released upon infection of the vulnerable hosts. Malware can cause significant damages, including consumption of network bandwidth, destructions of infected hosts, and leakage of private information, such as credit card numbers, etc.

Typical Malware propagation consists of a number of steps:

1. *Reconnaissance*: search vulnerable victim hosts by performing port scans;
2. *Infection*: transmit malicious payloads, exploit vulnerabilities on victim hosts to gain control;
3. *Discovery*: perform information-gathering activities on victim hosts, e.g., steal passwords and personal files;
4. *Destruction*: perform destructive activities on victim hosts, e.g., re-format their hard disks.

After the Infection step is done, the malware is ready to propagate from the newly infected host to another one by repeating the whole process. Note that not all malware propagation follow all of the above steps.

To perform a thorough port scan during reconnaissance, malware sends probe packets to each port on each victim host, and analyzes their responses. In a hypothetical scenario, a packet sent to FTP port 21 on a victim host triggers a reply packet, which is then analyzed by malware to infer detailed information, such as the type and version of the operating system, about the victim host. Based on this information, a well-tailored attack can be launched (e.g., exploiting the vulnerability that exists on the particular operating system).

Malware has to perform port scans for a huge number of IP address/port number combinations. In IPv4 networks, the size of the IP address space is 2^{32}, and the size of the port number space is 2^{16}. Hence, the size of the search space for the IP address/port number combination is 2^{48}. While the large size of the search space renders port scanning a daunting task, malware authors have employed sophisticated techniques to perform fast scanning. E.g., many real-world worms search vulnerabilities only on a particular port, which effectively reduces the size of the search space to 2^{32} [42].

It is clear that malware with different scanning and propagation strategies has different propagation time. A number of models have been proposed to characterize propagation of worms, including the state-of-the-art Analytical Active Worm Propagation (AAWP) model [18], and the epidemiological two-factor model [8]. Existing malware propagation models fail to consider a number of issues, including the following:

a) *That malware can scan a host for multiple vulnerabilities*: E.g., if malware fails to find any vulnerability on the FTP port 21 of a host, it can look for vulnerabilities on other ports, e.g., the DNS port 53. In case that malware discovers multiple vulnerabilities, it is able to exploit the most promising one according to some criteria (e.g., infection time).

b) *That scanning can be done by multiple threads*: Multi-threaded malware can scan and infect multiple machines concurrently. Moreover, since vulnerabilities exist

on many ports, multi-threaded scanning of multiple ports on one host is an effective way to speed up port scans. Most existing models, including the AAWP model, fail to consider that malware may spawn a large number of threads to scan concurrently.

c) *That exploitation of vulnerabilities and infection of victim hosts are not done instantly*: It takes time for malware to transmit its payload, exploit a vulnerability, and subvert the defense system on a victim host. A newly found vulnerable host can neither be infected immediately nor be ready right away to infect other hosts. Although the AAWP model claims to incorporate the infection time, it simply makes the clock ticks larger, without calculating the ratio of scan time to infection/propagation time. In AAWP, all infected hosts perform scanning activities at the next time tick (denote it as t and time tick length as L). Therefore, newly infected hosts that were infected between time (t, t + L) are treated equally: hosts infected near time t perform the same number of scans as those infected near time t + L. Such equal treatment is imprecise. It should be noted that port scans can be done much faster than infections. In the extreme case, Figure 1(c) in [18] assumes that the infection time could be as long as 60 seconds, while the scanning time for one IP/port combination is usually shorter than 0.1 second [19].

Theorem 1 in AAWP is proven by induction on the number of scans. If the scan is successful, it brings in a newly infected host. Hence, each induction step adds at most one host. At the next time tick, the number of infected hosts increases by at most one. AAWP assumes that the scans are performed step by step, i.e., in each step the scanning of one worm is performed, and the number of infected hosts is updated. The assumption differs from most real-world scenarios. For example, the famous NMAP scanner [19] is capable of scanning many hosts in parallel by dividing targets into multiple groups, and scanning an entire group at a time.

d) *That malware propagation can start from multiple places rather than a single starting point, and infected hosts can collaborate to increase damage (e.g., the Botnet [5] and the orchestrated attacks on Estonia [13])*: Multiple attackers can simultaneously release the same malware at multiple places. Researchers suspect that the

Witty Worm [33] was released from multiple IP addresses. Malware can be released in different geographical regions as well, e.g., Europe, Asia, and North America, to significantly expedite its propagation. Multiple starting points are not well-represented by existing models.

In summary, little was done to understand the effects of Multi-port scanning, Multi-threading, Infection time, Multiple starting points, and Collaboration (MMIMC) on malware propagation. In this research, we quantitatively measure the effects of MMIMC on infected hosts. We employ the Fibonacci Number Sequence (FNS) to model the effects of infection time. The extended model can explain the impact of threading, infection time, and multiple-attacker collaboration, as well as the effects of hitlist size, birth rate, and patching rate on malware propagation. We derive the Shift Property, which illustrates that different malware initializations can be represented by shifting their propagations on the time axis. We prove the Linear Property, which shows that the effects of multiple-attacker collaboration can be represented by linear combination of individual attacks. Experimental results show that the above issues significantly affect malware propagation and verify our analysis. To our knowledge, this is the first research that provides quantitative analysis and experimental results on the effects of MMIMC.

2.2 Related Work

Scan Strategy.

Over the years, researchers have proposed various scanning algorithms for malware, including:

(a) *naive random scanning*, in which malware chooses a random address niformly from the IP address space [18];

(b) *localized scanning*, in which malware scans a local IP address with a high probability p and scans a random address with a low probability (1-p) each time [3];

(c) *importance scanning*, in which malware assumes that the vulnerable hosts are unevenly distributed and such distributions are obtainable [32];

(d) *self-learning scanning*, in which malware estimates the distribution of the vulnerable hosts [26];

(e) *hit-list scanning*, in which malware uses an existing list. e.g., BGP routing table list, social network list, etc., to look for vulnerable hosts [24];

(f) *permutation scanning*, in which malware can determine whether a host is already infected and changes scan targets [24];

(g) *sampling scanning*, in which malware samples a target network before spreading to it [17]; and

(h) *passive scanning*, in which malware analyzes the network traffic passively.

Malware Propagation.

Wagner *et al.* [27] present characteristics of worms, including protocol, size of the payload, and scanning strategy, etc. Zou *et al.* [42] analyze the performances of different propagation strategies. Voyiatzis *et al.* [25] describe a class of worms that target network components such as routers. Vojnovic *et al.* [17] discuss how to minimize the required number of scans to infect hosts. Storm Worm [28], [29] uses the Distributed Hash Table (DHT) protocol based on Kademlia [16] to control infected nodes. Chen *et al.* [18] propose the Analytical Active Worm Propagation (AAWP) model. Zou *et al.* [8] propose the epidemiological two-factor model. Dagon *et al.* [11] discuss the taxonomy of Botnets.

2.3 Background on Fibonacci Number Sequence

In this section, we briefly summarize the Fibonacci Number Sequence (FNS) and discuss its generalizations. We infer several important properties of the FNS and discuss their uses in the malware propagation. In Section 2.4 we discuss in detail on how the FNS is applied to analyze the malware propagation and to model multi-threading, infection time, multiple start points, and collaborative attacks. The FNS is

Table 2.1: Notations used in this research

Notation	Explanation
b	the number of IP addresses on the blacklist of the malware
c	the number of ports scanned for each IP address
w	the number of contagious hosts that can infect other hosts
q	the probability that a given IP address/port combination will be discovered by at least one infected host
d	destruction rate: the number of destructed hosts over the number of infected hosts
k	the number of threads in the malware
p	patching rate: the rate at which the vulnerable machines are patched
r	birth rate: the rate at which the new vulnerable hosts joins the network
v	the number of vulnerable (excluding infected) host/port combinations
V	the number of vulnerable (including infected) host/port combinations
PT	Propagation Time
IT	Infection Time

named after Leonardo Fibonacci. Interested readers are referred to [14] for thorough discussions of the FNS. Table 2.1 lists the notations used in this research.

2.3.1 Fibonacci Rabbit Problem

We briefly present the famous Fibonacci rabbit problem: *In the beginning, there is no rabbit. A new pair of baby rabbits is introduced after one month. The baby rabbits get mature after one month. Each pair of mature rabbits has the ability to and will give birth to a new pair of baby rabbits every month. The question is: how many pairs of rabbits are there after n months?*

2.3.2 Definition of Fibonacci Number Sequence

To solve the fibonacci rabbit problem, we assume that rabbits never die. We use F_n to represent the number of pairs of rabbits there are after n months. Note that $F_0 = 0$ and $F_1 = 1$. We observe that $F_2 = 1 \neq 2$, since after 2 months the first pair of baby rabbits will get mature and cannot yet give birth to new baby rabbits. Note that in the malware propagation field, most existing models ignored this issue.

Since rabbits never die, to calculate how many pairs of rabbits there are after n (n>1) months, we simply add the newly born rabbits to the existing rabbits after (n-1) months, which is represented by F_{n-1}. Not all those F_{n-1} pairs of rabbits are mature. Because the baby rabbits take one months to get mature, we observe that the baby rabbits are those born within one month, i.e., the (n-2, n-1) month window. Therefore, the rabbits that were born before this window are all mature by Month n. There are F_{n-2} pairs of such rabbits.

Assume that F_{n-1} and F_{n-2} are known. We have:
$F_n = F_{n-1} + F_{n-2}$ when $n > 1$.

Hence, the solution can be summarized as:

$$F_n = \begin{cases} 0 & \text{if n = 0;} \\ 1 & \text{if n = 1;} \\ F_{n-1} + F_{n-2} & \text{if n > 1.} \end{cases} \tag{2.1}$$

We call the numbers generated by the recursive definition (2.1) Fibonacci numbers, and call the number sequence FNS.

2.3.3 Properties of Fibonacci Number Sequence

Closed-Form Expression

We can solve the recursive equation of the FNS with the initial conditions $F_0 = 0$ and $F_1 = 1$.

$$F_n = \frac{\phi^n - (1-\phi)^n}{\theta}, \quad where \theta \quad = \sqrt{5} \quad and \phi \quad = \frac{1+\theta}{2} \tag{2.2}$$

Since $|\frac{(1-\phi)^n}{\theta}|$ is a very small number (smaller than 0.1 when n is larger than 3), we can safely discard it and rewrite the result as:

$$F_n = \frac{\phi^n}{\theta}, \quad where \theta \quad = \sqrt{5} \quad and \phi \quad = \frac{1+\theta}{2} \tag{2.3}$$

Note that ϕ is the golden ratio (approximately 1.618).

Growth Rate

The growth rate of the FNS, regardless of the initial values (except for $F_0 = F_1 = 0$), is:

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi$$

Hence, the FNS approximately follows the exponential growth at the rate of the golden ratio ϕ when n is large. Note that the malware propagation is also exponential before saturation [28], [3].

2.3.4 Generic Fibonacci Number Sequence: Arbitrary Initialization

Definition of Generic Fibonacci Number Sequence

If the initial values of the FNS are changed to x and y respectively, we will have the generic FNS $G_{x,y,n}$:

$$G_n = \begin{cases} x & \text{if n} = 0; \\ y & \text{if n} = 1; \\ G_{n-1} + G_{n-2} & \text{if n} > 1. \end{cases} \tag{2.4}$$

How to Calculate Generic Fibonacci Number Sequence

$G_{x,y,n}$ can be represented by the original FNS.

$$G_{x,y,n} = xF_{n-1} + yF_n \tag{2.5}$$

Due to space limitations, we omit the proof for Equation (2.5). Note that $F_{-1} = F_1 - F_0$, thus Equation (2.5) still holds when n is 0.

Similar to Section 2.3.3, when n is larger than 3, we apply Equation (2.3) and rewrite $G_{x,y,n}$ as:

$$\begin{aligned} G_{x,y,n} &= xF_{n-1} + yF_n \\ &= x\frac{\phi^{n-1}}{\theta} + y\frac{\phi^n}{\theta} \\ &= (\frac{x}{\phi} + y)\frac{\phi^n}{\theta} \\ &= (\frac{x}{\phi} + y)F_n \end{aligned} \tag{2.6}$$

We observe that the Generic FNS can be approximately calculated by multiplying the original FNS by a constant factor.

The Shift Property of Generic Fibonacci Number Sequence

Given Equation (2.6) we can infer the "Shift" property of the Generic FNS, i.e., the generic Fibonacci number $G_{x,y,n}$ can be represented by the original Fibonacci number of F_{n+s}, where s is the number of shifts and s is equal to $\frac{\log(y\phi+x)}{\log\phi} - 1$. Formally:

Theorem 1 $G_{x,y,n} = F_{[n+\frac{\log(y\phi+x)}{\log\phi}-1]}$

Proof: According to Equation (2.6),

$$\begin{aligned} G_{x,y,n} &= (\frac{x}{\phi} + y)\frac{\phi^n}{\theta} \\ &= (\frac{x+y\phi}{\phi})\frac{\phi^n}{\theta} \\ &= (\frac{\phi^{\log_\phi(x+y\phi)}}{\phi})\frac{\phi^n}{\theta} \\ &= (\phi^{[\frac{\log(x+y\phi)}{\log\phi}-1]})\frac{\phi^n}{\theta} \\ &= \frac{\phi^{[n+\frac{\log(x+y\phi)}{\log\phi}-1]}}{\theta} \\ \textit{Apply} \quad \textit{Equation}(2.3), &= F_{[n+\frac{\log(y\phi+x)}{\log\phi}-1]} \end{aligned}$$

■

Given x and y, the number of shifts s is a constant number. Theorem 1 has important implications on the Fibonacci malware propagation: it quantifies the effects of different initialization values, and proves that the same effects can be achieved by "shifting" the index of the original FNS by a constant number. Hence, the effects of hitlist scanning and flash scanning, etc., can be quantified in the model by shifting the regular scanning. We discuss this further in Section 2.4.

The Linear Property of Generic Fibonacci Number Sequence

The Linear Property of the Generic FNS states that the sum of two Generic FNSes with initial values (x_1,y_1) and (x_2,y_2) is equivalent to the Generic FNS with the initial values ($x_1 + x_2$, $y_1 + y_2$, respectively. Formally:

Theorem 2 $G_{x_1,y_1,n} + G_{x_2,y_2,n} = G_{x_1+x_2,y_1+y_2,n}$

Proof: According to Equation (2.6),

$$\begin{aligned} G_{x_1+x_2,y_1+y_2,n} &= (\frac{x_1+x_2}{\phi} + y_1 + y_2)\frac{\phi^n}{\theta} \\ &= (\frac{x_1}{\phi} + y_1)\frac{\phi^n}{\theta} + (\frac{x_2}{\phi} + y_2)\frac{\phi^n}{\theta} \\ &= G_{x_1,y_1,n} + G_{x_2,y_2,n} \end{aligned}$$ ■

Corollary 1 $G_{mx,my,n} = mG_{x,y,n}$

Proof: According to Theorem 2,

$$\begin{aligned} G_{mx,my,n} &= G_{x,y,n} + G_{(m-1)x,(m-1)y,n} \\ &= 2G_{x,y,n} + G_{(m-2)x,(m-2)y,n} \\ &= ... \\ &= kG_{x,y,n} + G_{(m-k)x,(m-2)y,n} \\ &= ... \\ &= mG_{x,y,n} \end{aligned}$$ ■

2.3.5 Generic Lucas Number Sequence

A further generalization of the FNS is the Generic Lucas Number Sequence (LNS). Given constant integers x and y, we have:

$$H_n = \begin{cases} x & \text{if n = 0;} \\ y & \text{if n = 1;} \\ \alpha H_{n-1} - \beta H_{n-2} & \text{if n > 1.} \end{cases} \tag{2.7}$$

The Generic FNS is a special case of the Generic LNS when $\alpha = 1$ and β = -1. To investigate malware propagation, we are interested in the case where $\alpha = 1$ and β = -q ($|q|<\frac{1}{4}$). We discuss this further in Section 2.4.2.

When $\alpha = 1$ and β = -q ($|q|<\frac{1}{4}$), as in Section 2.3.3, we can solve the recursive equation of the special LNS with the initial conditions H_0 = x = 0 and H_1 = y = 1, and get its closed-form expression:

$$H_n = \frac{\phi^n - (1-\phi)^n}{\theta}, where\theta \quad = \sqrt{1+4q} \\ and\phi \quad = \frac{1+\theta}{2} \tag{2.8}$$

Since $|q|<\frac{1}{4}$ and 4q<1, we can expand θ using binomial expansion:

$$\begin{aligned} \theta &= \sqrt{1+4q} \\ &= \sum_{m=0}^{+\infty} \frac{(-1)^n(2n)!}{(1-2n)(n!)^2 4^n}(4q)^m \\ &\approx \sum_{m=0}^{n} \frac{(-1)^n(2n)!}{(1-2n)(n!)^2 4^n}(4q)^m, n = 2 \\ &= 1 + \frac{4q}{2} - \frac{(4q)^2}{8} \\ &= 1 + 2q - 2q^2 \end{aligned}$$

Hence, given that $|q|<\frac{1}{4}$, $|\frac{(1-\phi)^n}{\theta}| = |\frac{(\phi-1)^n}{\theta}| = |\frac{(\frac{\theta-1}{2})^n}{\theta}| = |\frac{(q-q^2)^n}{1+2q-2q^2}|$ is a very small number. Thus, we can can safely discard it and rewrite the result as (when $\alpha = 0$ and β = -1):

$$H_n = \frac{\phi^n}{\theta}, \quad where\theta \quad = \sqrt{1+4q} \quad and\phi \quad = \frac{1+\theta}{2} \tag{2.9}$$

Note that when q = 1, we get the closed-form expression for FNS as in Section 2.3.3. We observe that Equation 2.3 and Equation 2.9 differ only in the constants. Therefore, the properties, including the Shift Property and the Linear Property, of the Generic FNS all hold for the Generic LNS when $\alpha = 1$ and β = -q (q<$\frac{1}{4}$). Due to space limitations, we omit the formal proof for this observation.

2.4 Analysis of MMIMC and the Generic Fibonacci Malware Propagation (GFMP) Model

In this section, we extend the existing malware propagation models to address the issues of MMIMC.

2.4.1 Preliminaries

Probability on Port Scanning

We assume that during the reconnaissance step malware performs port scanning to discover vulnerable ports on the target host.

Malware can scan a part of all IP addresses. For instance, reverse engineering [3], [9] shows that Code Red I and II never scan local (127.0.0.0/8) and multicast (224.0.0.0/8) addresses. This is overlooked by researchers (e.g., in [8] the authors assume that CodeRed scans all IP addresses with equal probability). Assume that IPv4 is in use and malware puts b IP addresses on its blacklist, i.e., it never scans those IP addresses. Thus, the number of IP addresses malware scans is (2^{32} - b). Assume that malware scans c ports for each IP address. The size of the search space for malware is $c(2^{32}$ - $b)$.

While real-world scanners are mostly multi-threaded [19], existing malware propagation models overlook multi-threading issues. We assume that malware employs multi-threaded programming and scans multiple address/port combinations concurrently. If there are k threads for each malware scanning module, we assume that each infected host can scan k address/port combinations simultaneously.

We need to calculate how many new vulnerable IP address/port combinations are discovered in each time tick. Note that vulnerability discovery is not equivalent to successful infection. After the vulnerability discovery, malware needs time to propagate to victim hosts and exploit the vulnerability. Assume that there are v_i uninfected vulnerable IP address/port combinations (multiple ports on one host can be infected) at time tick i (time steps are equally sized). Given that each infected host can perform k scans simultaneously, we can calculate how many out of those v_i combinations can be discovered by all infected hosts.

Denote the number of newly infected hosts as n_i, the number of contagious hosts as w, and the probability that a given IP address/port combination is discovered

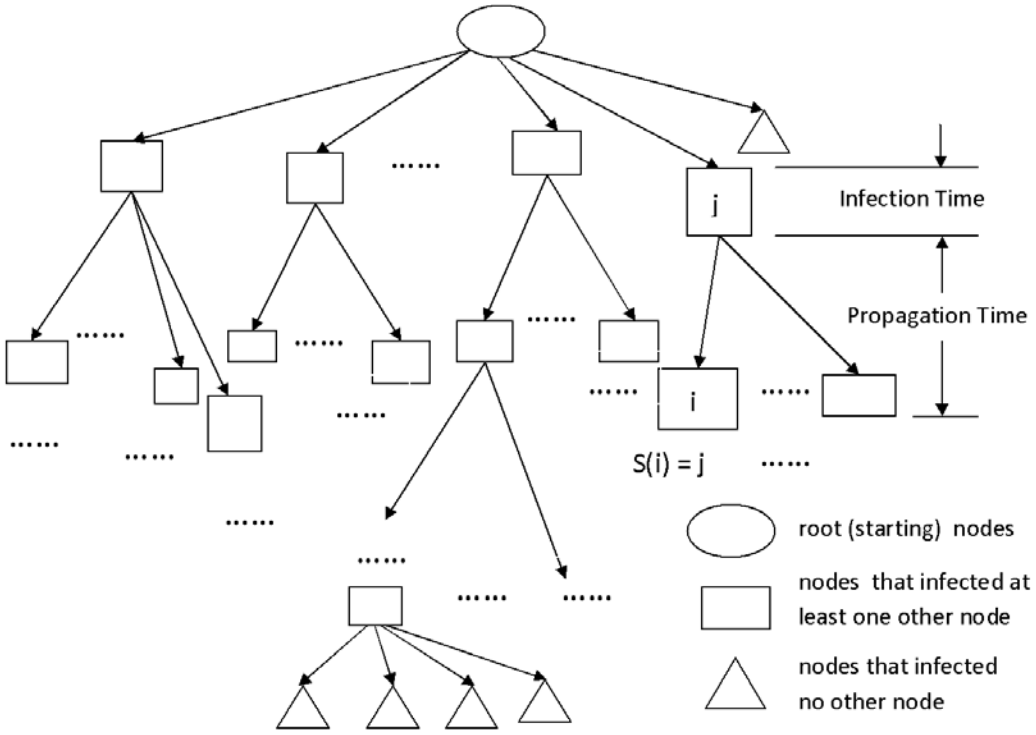

Figure 2.1.: The malware propagation tree.

by at least one infected host as q. Note $q = \frac{k}{c(2^{32}-b)}$. For a given IP address/port combination, we have:

P(discovered by at least one infected host at time tick i)

$$
\begin{aligned}
&= 1 - P(not\_discovered\_by\_any\_of\_w\_infected\_hosts) \\
&= 1 - P(not\_discovered\_by\_one\_infected\_host)^w \\
&= 1 - (1 - P(discovered\_by\_one\_infected\_host))^w \\
&= 1 - (1-q)^w
\end{aligned}
$$

Hence,

$$n_i = [1 - (1-q)^w]v_i \tag{2.10}$$

The Propagation Tree of Self-Propagating Malware

Assume that malware propagation starts from a single node. As shown in Fig. 2.1, the propagation tree of malware $PropT_r$ consists of:

(a) a *root node r*: the node where the malware executor releases it;

(b) *intermediate nodes*: nodes that caused direct infections of one or more nodes; and

(c) *leaf nodes*: nodes that caused no direct infections of other nodes.

Formally, we define:

(1) *Source (Parent) Function S*, such that:

S(i) = j, iff node i, j $\in PropT_r$, and j is a parent of i in the tree $PropT_r$. As shown in Fig. 2.1, if S(i) = j, node i is the child of node j.

(2) *Malware Propagation Tree* $PropT_r$, a directed tree in which each node is either:

(a) *root node r*, where $\nexists$ node j $\in PropT_r$ such that j = r and S(r) = j;

(b) *intermediate node i*, where $\exists$ node j $\in PropT_r$ such that j = i and S(j) = i; or

(c) *leaf node e*, where $\nexists$ node j $\in PropT_r$ such that j = e and S(j) = e, and $\exists$ node k $\in PropT_r$ such that k = e and S(e) = k.

The Propagation Forest of Self-propagating Malware

If malware is released at k sources, r_i, i $\in$ [1, ..., k], we can generate one propagation tree for the malware propagation rooted at each source node. The propagation forest of self-propagating malware F_{prop} is the disjoint union of the propagation trees rooted at nodes r_i, i $\in$ [1, ..., k], formally:

$$F_{prop} = \bigcup_{i=1}^{k} PropT_{r_i} \tag{2.11}$$

The Infection Time and Propagation Time

As shown in Fig. 2.1, there is a short delay between the intrusion of the malware and its propagation to other hosts. Such delay includes the time spent on the

vulnerability exploitation and subversion of the victim host. We denote the delay as *infection time*.

Intuitively, we define the *Infection Time (IT)* as the time interval between the start of the infection on a particular host (i.e., the time when the host was initially intruded) and the start of propagation on the same host (i.e., the time when the same host was starting to infect other hosts). Formally,

$$IT = T_{StartPropagation} - T_{StartInfection} \tag{2.12}$$

Actual infection times may vary and follow particular probability distributions.

We could define the *Propagation Time (PT)* between two hosts as the time interval between the infection of a particular host (denote it as s) and the successful infection of a subsequent target host (denote it as T(s)) that was caused by this particular host. Formally,

$$PT(s, T(s)) = T_{Infection(T(s))} - T_{Infection(s)} \tag{2.13}$$

For a *host m* that was never intruded or infected successfully, the time of infection ($T_{Infection(m))}$) is defined as $+\infty$ (infinite).

We can measure the propagation time for all infected hosts and collect statistics about them. E.g., we can calculate the average propagation time. There is one problem with definition (2.12): it works only if there is at least one subsequent successful infection from the original host (s). If such infection was unsuccessful (e.g., if the target host was invulnerable) or there was no subsequent infection attempt (e.g., if malware on the host was quarantined by administrators) the propagation time is $+\infty$ (infinite).

Alternatively, we can calculate the propagation time from the infected hosts, under the observation that each infected host must be infected by some source host. Hence, we define the *Propagation Time (PT)* between a host and its infector as: the time interval between the successful infection of a particular host (denote it as t) and

the infection of the host that infected t [15] (denote it as S(t)). Formally,

$$PT(S(t), t) = T_{Infection(t)} - T_{Infection(S(t))} \tag{2.14}$$

We can infer several important properties for Propagation Time (PT).

1. Additivity: if there are three hosts (Host m, n, and o) that satisfy the following two conditions:

a) the malware propagated directly from Host m to n; and

b) the malware propagated directly from Host n to o,

then the propagation time between Host m and o is the sum of the propagation time between Host m and n and the propagation time between Host n and o. Formally,

For Hosts m, n, and o that satisfy S(o) = n and S(n) = m:

$$PT(m, o) = PT(m, n) + PT(n, o), \tag{2.15}$$

2. Diameter: the diameter of the tree (Diameter($PropT_r$))is the time elapsed since the release of the malware until the infection of the last vulnerable hosts (denote it as lv). Hence, we can use PT(r, lv) to represent the diameter of the tree: Diameter($PropT_r$) = PT(r, lv).

Assume that there are n intermediate nodes on the path between r and lv. We denote them as $node_i$, i $\in$ [1..n], where:

$$\begin{cases} S(node_i) = r, & \text{if i = 1;} \\ S(lv) = node_i, & \text{if i = n;} \\ S(node_{i+1}) = node_i & \text{if i } \in \text{ (1..n).} \end{cases} \tag{2.16}$$

Using Property 1, Diameter(T_{prop}) can be further calculated as:

Diameter($PropT_r$)

= PT(r, lv)

$= \mathrm{PT}(\mathrm{r}, node_i) + \mathrm{PT}(node_1, node_2) + ... + \mathrm{PT}(node_i, node_{i+1}) + ... + \mathrm{PT}(node_n, \mathrm{lv})$

$$= PT(r, node_i) + PT(node_n, lv) + \sum_{\substack{i \\ \mathbf{i} \in (\mathbf{1}..\mathbf{n})}} PT(node_i, node_{i+1}) \tag{2.17}$$

2.4.2 Generic Fibonacci Malware Propagation (GFMP) Model

We employ Fibonacci Number Sequence (FNS) to model infection time. Recall that in the Fibonacci rabbit problem, newly-born rabbits cannot give birth to baby rabbits immediately. Instead, they need some time to mature, which is reminiscent of the infection/propagation time problem discussed above: a captured host cannot scan and infect other hosts until its infection matures, i.e., until it is completely infected.

1) Recursive Equation for the Malware Propagation

Denote the number of all vulnerable hosts in the beginning as V and the number of infected hosts as I_j, where j denotes the time tick. Denote the length of the time slice between time ticks as L (one time slice could represent one second). Assume that the administrators may patch the vulnerable hosts. Assume that the propagation time is two time slices for all infections. Hence, the newly infected hosts intruded at time t are not able to infect new hosts at time $t + L$, but will be able to infect new hosts at time $t + 2L$. At time tick $j + 2$, there are I_j infected hosts that are contagious and can infect other hosts. Formally:

$$w = I_j \tag{2.18}$$

At time tick $j + 1$, the number of uninfected vulnerable hosts is the number of all unpatched vulnerable (including infected and newly born) hosts minus the number of infected vulnerable hosts. Assume that malware can carry destructive payloads (e.g., programs that can re-format the hard drive). In this case, the destructed hosts are wiped out and removed from the vulnerable host list. Note that neither dead (or significantly damaged) nor newly-born hosts could be patched.

We define *destruction rate* as the number of destructed hosts divided by the number of infected hosts, considering that only infected hosts can be destroyed. Therefore,

we calculate the number of dead hosts by multiplying the destruction rate by the number of infected hosts, instead of the number of all vulnerable hosts. We denote the destruction rate of the hosts as d, the birth rate of the vulnerable hosts (e.g., new vulnerable hosts that just joined the network) as r, and the patching rate of infected hosts as p. Formally, the number of hosts that are vulnerable (including infected and newly-born) and can be patched at time tick $j+1$ is:

$$v_{j+1} = (1-p)v_j - dI_j + rv_j = (1-p+r)v_j - dI_j$$

This is a recursive equation. We expand the recursion and get:

$$v_{j+1} = (1-p+r)^{j+1}v_0 - \sum_{k=0}^{j}(1-p+r)^k dI_{j-k}.$$

Given that $v_0 = \mathrm{V}$, we have:

$$v_{j+1} = (1-p+r)^{j+1}V - \sum_{k=0}^{j}(1-p+r)^k dI_{j-k} \tag{2.19}$$

The number of hosts that are vulnerable but not infected is:

$$v'_{j+1} = v_{j+1} - I_{j+1} \tag{2.20}$$

After one time slice (time tick $j+2$), without considering destruction and patching, the number of infected hosts is the sum of the number of infected hosts at the previous time tick $(j+1)$ and the number of newly infected hosts during the time slice. The number of infected hosts that died or were patched during the time slice is

$$dp_{j+1} = (d+p)I_{j+1} \tag{2.21}$$

The number of newly infected hosts is calculated in Section 2.4.1.

Given (2.10), (2.18), (2.19), (2.20), (2.21), we have:

$$\begin{aligned} & I_{j+2} \\ &= I_{j+1} + n_{j+1} - dp_{j+1} \\ &= I_{j+1} + v_{j+1}(1-(1-q)^{I_j}) - (d+p)I_{j+1} \\ &= (1-d-p)I_{j+1} + [(1-p+r)^{j+1}V - \\ & \sum_{k=0}^{j}((1-p+r)^k dI_{j-k}) - I_{j+1}][1-(1-q)^{I_j}] \end{aligned} \tag{2.22}$$

Note this recursive growth function applies when there is at least one vulnerable host.

2) Special Cases

Special cases are as follows:

a) If the birth and patching rates are equal, (2.22) can be simplified to:

$$\begin{aligned} & I_{j+2} \\ &= (1-d-p)I_{j+1} + \\ & (V - d\sum_{k=0}^{j} I_{j-k} - I_{j+1})[1-(1-q)^{I_j}] \end{aligned} \tag{2.23}$$

b) If the birth, destruction, and patching rates are all zero, (2.22) can be simplified to:

$$I_{j+2} = I_{j+1} + (V - I_{j+1})[1-(1-q)^{I_j}] \tag{2.24}$$

c) Binomial expansion can be used to expand and simplify 1 - (1 - q)^{I_j}:

$$\begin{aligned} & 1-(1-q)^{I_j} \\ &= 1 - \sum_{m=0}^{I_j} \binom{I_j}{m} (-q)^m \end{aligned} \tag{2.25}$$

We observe that: k represents the multi-threading level of the malware propagation scanner, and normally ranges from 1 to 2^{10} or one thousand; V represents the number of vulnerable hosts(including infected hosts), and is normally smaller than 2^{20} or one million; c represents the number of ports that the malware is scanning, and $c > 0$; and b represents the number of IP addresses that the malware puts on the blacklist.

If the malware puts local and multicast addresses on the blacklist only, 2^{32} - $b \approx 2^{32}$. Hence, $qV = \frac{kV}{c(2^{32}-b)} < \frac{2^{10}\times 2^{20}}{2^{32}} = \frac{1}{4}$. Note that these are conservative estimations since normally k is much smaller than 2^{10} and V is smaller than 2^{20}. Since the number of infected hosts cannot be larger than the number of all vulnerable hosts, i.e., $I_j \leq$ V, we conclude that qI_j is small. Therefore, we can safely discard the high order elements in Equation 2.25. We can rewrite Equation (2.24) as:

$$
\begin{aligned}
I_{j+2} &= I_{j+1} + (V - I_{j+1})[1 - (1-q)^{I_j}] \\
&= I_{j+1} + (V - I_{j+1})[1 - \sum_{m=0}^{1} \binom{I_j}{m} (-q)^m] \\
&= I_{j+1} + (V - I_{j+1}) \binom{I_j}{1} (q) \\
&= I_{j+1} + qI_j(V - I_{j+1}) \\
&= I_{j+1} + qI_jV - qI_jI_{j+1}
\end{aligned}
$$

During the initial phase of the spread of the malware, $\frac{I_{j+1}}{V}$ is a small number, so we can safely throw away $-qI_jI_{j+1}$. Therefore:

$$I_{j+2} = I_{j+1} + qVI_j \tag{2.26}$$

Equation 2.26 suggests that the initial spread of the malware approximately follows the Generic Lucas Number Sequence (LNS) [14] with $\alpha = 1$ and $\beta =$ -qV:

$$I_j = \begin{cases} x & \text{if j = 0;} \\ y & \text{if j = 1;} \\ I_{j-1} - (-qV)I_{j-2} & \text{if j > 1.} \end{cases} \tag{2.27}$$

d) We now discuss the effects of different lengths of the propagation time. Equation 2.26 holds when the propagation time is $2L$ (twice as much as the length of the unit time slice). Generally, if propagation time is eL, where e is an integer, we have:

$$I_{j+2} = I_{j+1} + qVI_{j+2-e}, \quad where \quad j + 2 > e \tag{2.28}$$

We now have the equation that quantitatively measure the effects of propagation time. The equation shows that the longer propagation time is, the slower the malware propagates, which follows the intuition that longer propagation time hampers malicious activities of newly infected hosts.

2.4.3 Properties of the GFMP Model

We use the GFMP model to study the issues of threading, infection time, multiple starting points, and collaborations. Due to space limitations, we omit the discussion of properties of FNS and use them directly. Interested readers are referred to [30] for details.

Multi-threading and the Closed-Form Expression

The closed-form expression for the number of infected hosts at time tick j, when $x = 0$ and $y = 1$, is:

$$I_j = \frac{\phi^j}{\theta}, \quad where\theta \quad = \sqrt{1+4qV} \quad and\phi \quad = \frac{1+\theta}{2}$$
$$= \frac{[\sqrt{c(2^{32}-b)}+\sqrt{c(2^{32}-b)+4kV}]^j}{2\sqrt{c(2^{32}-b)+4kV}[2\sqrt{c(2^{32}-b)}]^{j-1}}$$

Note that the malware propagation stops when all vulnerable hosts that can be infected are infected. Hence, during the propagation $I_j \leq V$. Hence, we can rewrite I_j as:

$$I_j = \begin{cases} \lambda, & \text{if } \lambda \leq \text{V}; \\ V, & \text{if } \lambda > \text{V}. \\ & (\lambda = \frac{[\sqrt{c(2^{32}-b)}+\sqrt{c(2^{32}-b)+4kV}]^j}{2\sqrt{c(2^{32}-b)+4kV}[2\sqrt{c(2^{32}-b)}]^{j-1}}) \end{cases} \tag{2.29}$$

In Equation 2.29, k denotes the number of active threads in the malware scanner. As k increases, the infection rate increases. However, note that multi-threaded programs can easily generate huge network traffic by sending out a large number of packets. While context switching for threads are smaller than those of processes, the

costs increase as k increases. Real-world multi-threaded malware normally employs 10 - 100 threads. Equation 2.29 was derived from Equation 2.26, where we assume that the number of previously infected hosts is much smaller than the number of all vulnerable hosts ($\frac{I_{j-1}}{V}$ is small), and dropped $-\frac{k}{c(2^{32}-b)}I_jI_{j+1}$. Hence, Equation (2.29) grows faster than actual malware propagation when the number of infected hosts is large. Experimental results that support Equation 2.29 are discussed in Section 3.5.

Sophisticated Scanning and the Shift Property

In Section 2.4.3, we derived the closed-form expression when malware employs multi-threaded random scanning, and the initial values of x and y are 0 and 1, respectively. However, malware can employ more sophisticated scanning techniques, such as a combination of scanning strategies. Malware can use hitlist scanning to infect a large number of pre-selected vulnerable hosts [24] before performing regular random scanning on newly infected hosts. Our extended model represents such scanning strategies by different initializations of x and y. E.g., if the size of the hitlist is h, we assume that at time tick 1 the number of infected hosts is h (the original release point of malware) instead of 1, i.e., $x = 0$ and $y = h$.

According to the Shift Property of FNS, The Generic LNS sequence determined by Equation 2.29 with initial values x and y can be calculated as:

$$GI_{x,y,j} = I_{[j+\frac{\log(y\phi+x)}{\log\phi}-1]} \tag{2.30}$$

When $x = 0$ and $y =$ h, we have:

$$GI_{0,h,j} = I_{[j+\frac{\log(h\phi)}{\log\phi}-1]} \tag{2.31}$$

Hence, the number of infected hosts of the malware with a hitlist of size h and the combined scanning strategy at time j can be represented by the number of infected hosts of the original random-scanning malware at time $(j + s)$, where s is the shifting number $\frac{\log(h\phi)}{\log\phi} - 1$.

Furthermore, according to properties of the FNS,

$$GI_{x,y,j} = xI_{j-1} + yI_j$$

Hence, the propagation of malware employing the combined hitlist and random scanning is the linear combination of two propagations of malware employing random scanning only. When $x = 0$ and $y = h$, we have:

$$GI_{0,h,j} = hI_j \tag{2.32}$$

We call h the linear Fibonacci Coefficient (FC) of the linear combination.

Multiple Starting Points, Collaborative Attacks and the Linear Property

Malware propagation can start from multiple places in the network rather than from a single point, and infected hosts can collaborate with each other to cause much more damage. E.g., the coordinated Botnet zombie nodes can collaborate to launch DoS attacks [5], and the well-orchestrated collaborative attacks on Estonia caused large-scale disruptions [13].

We consider the representation of the following attacks:

Case 1. There are m uncoordinated attackers who release the same copy of malware at m places simultaneously. We assume that malware employs the localized random scanning strategy. We assume that the search spaces of attackers are independent (e.g., attackers divide the whole IP address space equally into m parts and each attacker will be responsible for one part). For the initializations, we assume that $x = 0$ and $y = 1$ for all attackers. According to Equation 2.27, the propagation of malware released by all attackers can be represented as $I_{0,1,j}$ because their initial values and β coefficients are the same. Note that $|\beta| = \frac{kV}{c(\frac{2^{32}}{m} - b)}$ now since the search space for each attacker is now reduced to $\frac{2^{32}}{m}$. Recall that we have $|\beta| < \frac{1}{4}$. As discussed in Section 2.4.2, if we assume that $V = 2^{20}$ and $b = 0$, we have $\frac{kV}{c(\frac{2^{32}}{m} - b)} = \frac{mk}{2^{12}c} < \frac{1}{4}$. Hence, $\frac{mk}{c} < 2^{10}$, which means that the product of the number of threads per malware and the number of attackers divided by the number of scanned ports is smaller than

1024, if there are one million vulnerable hosts. We assume that this condition holds and denote the propagation of the whole collaborative attack as $ITOTAL_{x_{total},y_{total},j}$.

According to the Linear Property of the FNS, we have:

$$\begin{aligned} ITOTAL_{x_{total},y_{total},j} &= \sum_{n=0}^{m-1} I_{0,1,j} \\ &= mI_{0,1,j} \\ &= I_{0,m,j} \end{aligned}$$

Hence, the number of infected hosts of m uncoordinated attacks that perform localized scanning is equivalent to that of the single attack released at one point with initial values $x_{total} = 0$, and $y_{total} = m$.

Case 2. There are still m collaborative attackers releasing malware. We assume that malware employs the sophisticated scanning strategy (but each malware copy shares the same search space) and malware at different hosts can communicate with each other to avoid duplicate infection attempts. Note we do not assume that infected hosts can avoid duplicate scanning (in which multiple attackers can be modeled as one attacker with a huge number of threads and minimal thread maintenance costs). We assume that initial values of the propagation of the malware released by Attacker A_n are x_n and y_n (n$\in$[0, ..., m)). We still denote the propagation of the whole collaborative attack as $ITOTAL_{x_{total},y_{total},j}$.

According to Linear Property of the FNS, we have:

$$\begin{aligned} ITOTAL_{x_{total},y_{total},j} &= \sum_{n=0}^{m-1} I_{x_n,y_n,j} \\ &= I_{\sum_{n=0}^{1} x_n,\sum_{n=0}^{1} y_n,j} + \sum_{n=2}^{m-1} I_{x_n,y_n,j} \\ &= \dots \\ &= I_{\sum_{n=0}^{m-1} x_n,\sum_{n=0}^{m-1} y_n,j} \end{aligned} \tag{2.33}$$

Hence, the power of the m collaborative attacks is equivalent to the single attack released at one point with initial values $x_{total} = \sum_{n=0}^{m-1} x_n$, and $y_{total} = \sum_{n=0}^{m-1} y_n$.

Equation 2.33 quantifies the power of collaborative attacks, and grows much faster than Equation 2.27.

2.5 Experiments

In this section, we present the experimental results on the impact of threading, infection time, and multiple-attacker collaboration, as well as the effects of hitlist size, birth rate, and patching rate on malware propagation. We have conducted the experiments on a network that consists of a Pentium 4 workstation and virtual machines. We simulate the worm propagation and use one machine to simulate multiple victim hosts. We implemented the malware propagation model in C++. Without loss of generality, in all the experiments, we set V (the number of all vulnerable hosts) to 1,000,000, c (the number of ports the malware scans for one host) to 1, and b (the number of IP addresses that the malware does not scan) to the size of local and multicast address space, which is approximately 2^{25}.

2.5.1 Verification of the GFMP Model: the Shift Property

We perform experiments to verify our theoretical GFMP model before employing it to study the effects of other parameters. In particular, we want to show the Shift Property discussed in Section 2.4.3. In this experiment, we set k to 100, d to 0, p to 0.0002, and r to 0.0002.

From Equation 2.32, $GI_{0,h,j} = hI_j$, the number of infected hosts with hitlist size h divided by the number of infected hosts with hitlist size 1 is h. Hence, if sizes of the hitlists are 2, 100, and 200, the quotients are 2, 100, and 200, respectively. Note that the number of infected hosts with hitlist size 200 is $\frac{200}{100}$=2 times of the number of infected hosts with hitlist size 100.

Fig. 2.2 shows the results on the propagation with hitlist sizes 1, 2, 100, and 200. Note that numbers of infected hosts for hitlist sizes 1 and 2 are enlarged 100 times. The results confirm our theoretical analysis, and verify the Shift Property.

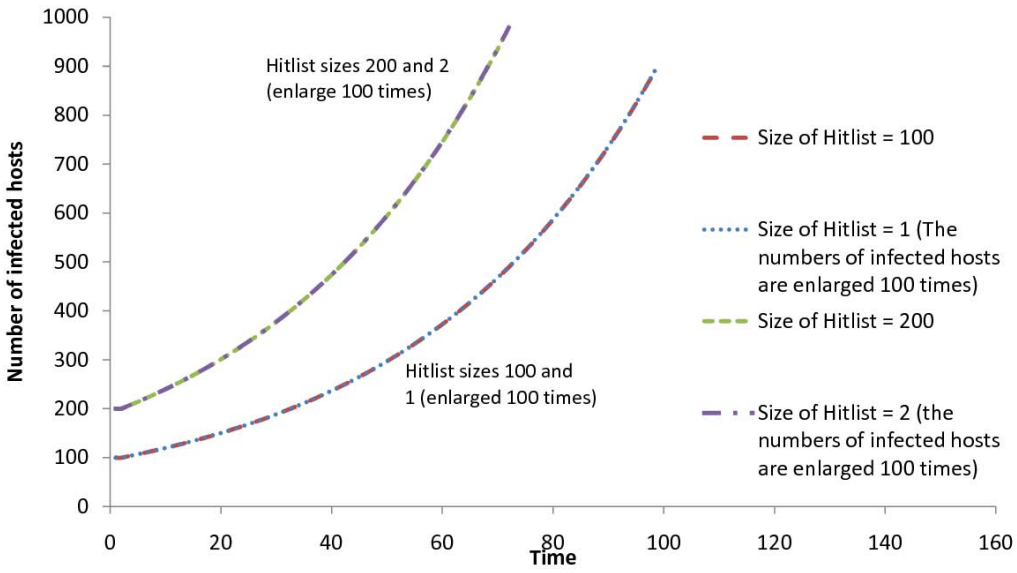

Figure 2.2.: The propagation of hitlist size 100 and 200.

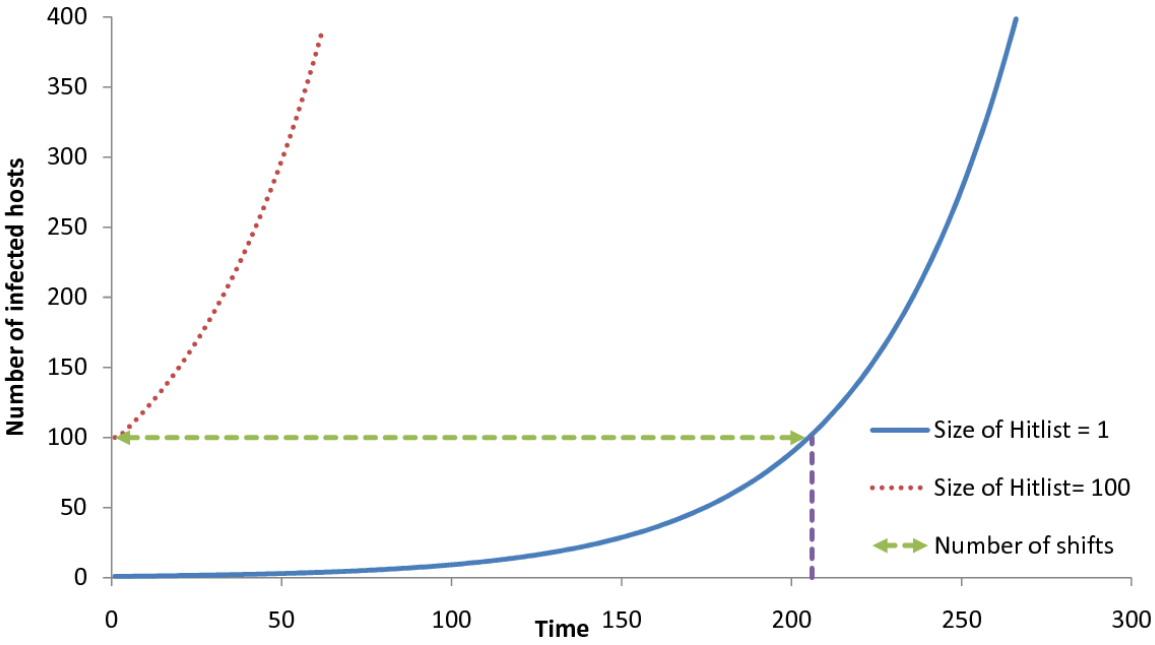

Figure 2.3.: The propagation of hitlist size 100 and 200.

For instance, numbers of infected hosts with hitlist size 100 (or 200) are essentially coincident with the numbers of infected hosts (enlarged 100 times) with hitlist size 1 (or 2). Numbers of infected hosts with hitlist size 200 are approximately twice as many as those with hitlist size 100.

According to Equation 2.31, $GI_{0,h,j} = I_{[j+\frac{\log(h\phi)}{\log\phi}-1]}$. Therefore, we can compute the number of shifts required to calculate the number of infected hosts with hitlist size h = 100. Since $k = 100$, $V = 2^{20}$, $b = 2^{25}$, $c = 1$, we have: $q = \frac{100*2^{20}}{1*(2^{32}-2^{25})} = \frac{100}{32*127}$ $= 0.025$. Hence, $\theta = \sqrt{1+4q} = \sqrt{1.098} = 1.048$, and $\phi = \frac{1+\theta}{2} = 1.024$ Therefore, the number of shifts is : $\frac{\log((h)\phi)}{\log\phi} - 1 = \frac{log(100*\phi)}{log\phi}$ -1 = $\frac{2.010}{0.010}$ - 1 = 200.

Fig. 2.3 confirms our theoretical analysis. The solid line shows the propagation with hitlist size 1. The dotted line shows the propagation with hitlist size 100. The dashed line that connects the solid and dotted lines illustrates the number of required shifts, which is approximately constant. The projection of the dashed line on the x-axis shows that the number of shifts is roughly equal to 200, which verifies our analytical result.

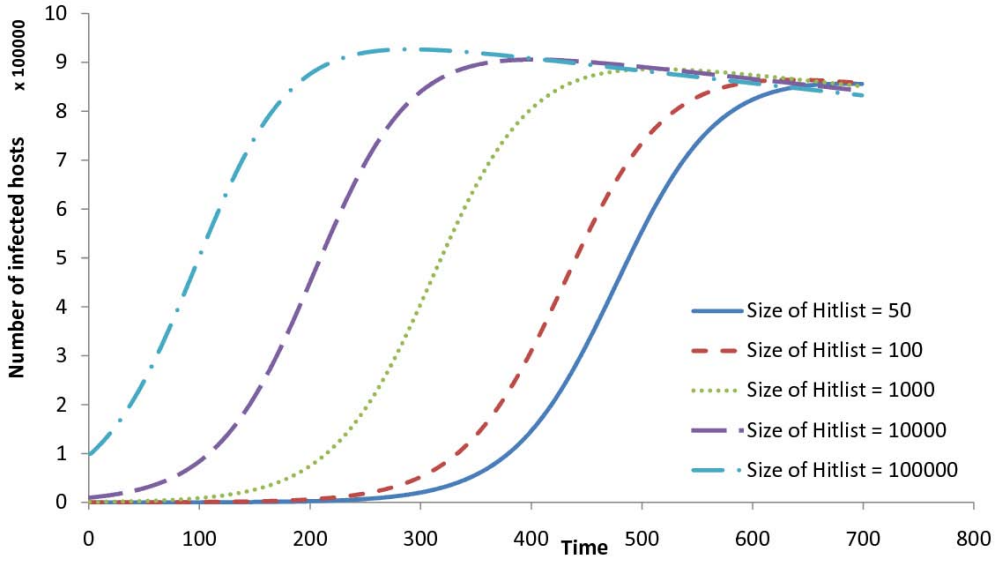

Figure 2.4.: The propagation of the multi-threaded malware.

2.5.2 The Effect of Different Hitlist Sizes on Multi-threaded Propagation

We have conducted experiments to evaluate whether the hitlist scanning can accelerate the propagation of multi-threaded malware, and compared the effects of different hitlist sizes on the malware propagation. In this experiment, we set k (the number of threads in the malware vulnerability scanner of the malware) to 100, d (destruction rate) to 0.0001, p (patching rate) to 0.0002, and r (birth rate) to 0. Note that we set birth rate to 0 to show the effects of threading (otherwise the increased number of infected hosts might be caused by newly joined vulnerable hosts).

Fig. 2.4 shows the malware propagation with hitlist sizes 50, 100, 1,000, 10,000, and 100,000. We observe that the propagation speed of the multi-threaded malware increases as the size of the hitlist increases. Specifically, with hitlists of sizes 100,000, 10,000, 1,000, 100, and 50, the malware propagated to 500,000 hosts in 100, 210, 319, 441, and 489 time ticks (seconds), respectively. We conclude that the hitlist scanning can effectively accelerate the multi-threaded malware propagation, especially when the size of the hitlist is large.

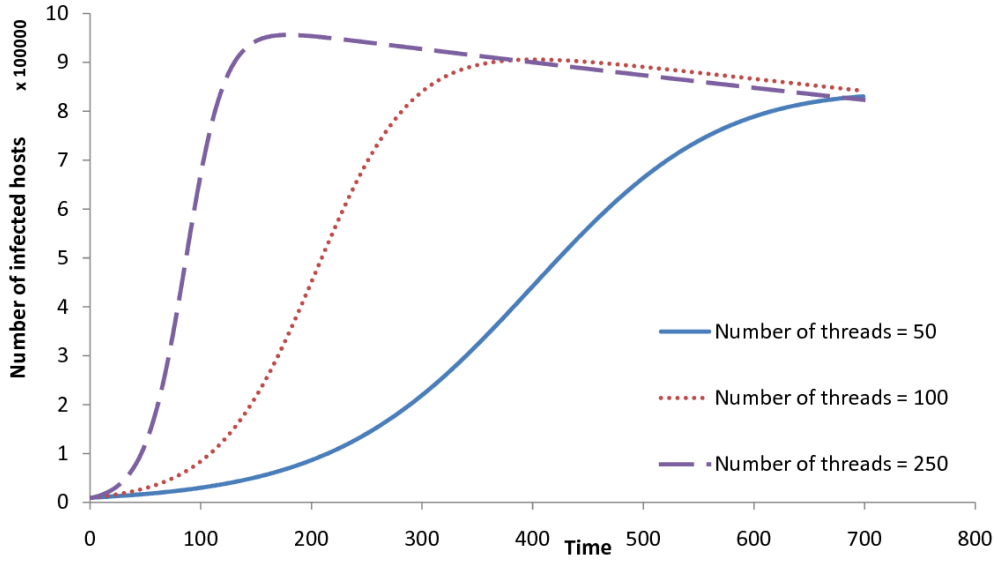

Figure 2.5.: The malware propagation with different number of threads.

When the size of the hitlist is 100,000, the malware propagation reached its peak after 290 time ticks, after which the actual number of infected hosts decreased. Such decrease is caused by patching and destruction. In our experiment, destruction and patching rates are not zero. Moreover, we set the birth rate to zero so that no new vulnerable hosts will join the network. Therefore, after the malware propagation reached its peak, there will be no new vulnerable host to infect, and patched hosts can no longer be infected. Hence, the number of infected hosts decreases. Note that different birth and patching rates can cause different propagation behavior. We discuss our experimental results on the birth patching rates in Sections 2.5.4 and 2.5.5, respectively.

2.5.3 The Effect of Different Threading-Levels

We conducted experiments to study how the number of scanning threads affects malware propagation. In this experiment, we set d to 0.0001, p to 0.0002, r to 0, and the hitlist size to 10000.

Fig. 2.5 shows the propagation with different threading-levels: 50, 100, and 250. We observe that the propagation speed increases as the number of threads in the malware increases. The effects of the multi-threads are significant: when the number of threads is 50, the malware took almost 700 time ticks to infect 800,000 hosts, while the same malware took approximately 300 time ticks with 100 threads and 100 time ticks with 250 threads to accomplish the same task. Note that when the number of threads is 250, the malware propagation reaches its peak in less than 200 time ticks. The number of infected hosts then decreased because of destruction and patching, since we assume that a patched host cannot be infected again in this experiment. The patching rate we set in this experiment is fairly high (0.0002), which means that in every time tick two out of one thousand infected hosts are patched.

2.5.4 The Effect of Different Birth Rates

We conducted experiments to study the effects of different birth rates on the malware propagation. In this experiment, we set d to 0.0005, p to 0.0000, k to 100, and the hitlist size to 10,000. Note we set the patching rate to 0 to focus on the birth rate.

Fig. 2.6 shows the propagation with birth rates 0, 0.0001, 0.0002, 0.0003, 0.0004, and 0.0005. Note that when the birth rate is 0.0005, it is equal to the destruction rate (0.0005). We observe that the birth rate does matter during the malware propagation. Specifically, we observe that the number of infected hosts peaked at 1,050,000 when the birth rate is 0.0004, while the number of the infected hosts peaked at only 917,000 when the birth rate is 0.

Moreover, when the birth rate is 0.0005, which is equal to the destruction rate, we observe that the number of infected hosts peaked at 1,080,000. The number of infected hosts neither increased nor decreased afterwards. Therefore, an equilibrium was reached: although 5 out of 10,000 hosts are destructed in each time slice, 5 out of 10,000 hosts are newly born and infected by the malware in each time slice.

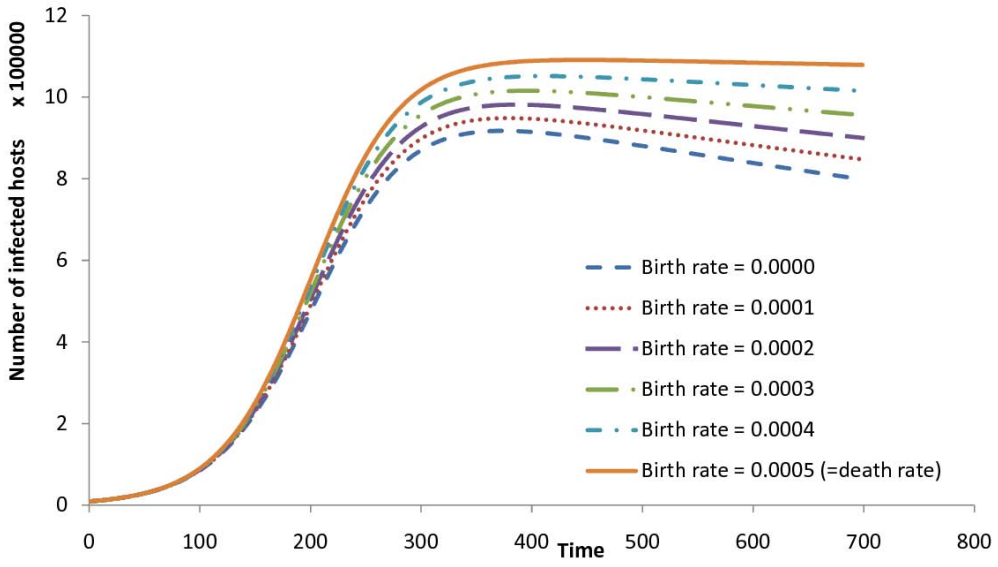

Figure 2.6.: The malware propagation with different birth rates.

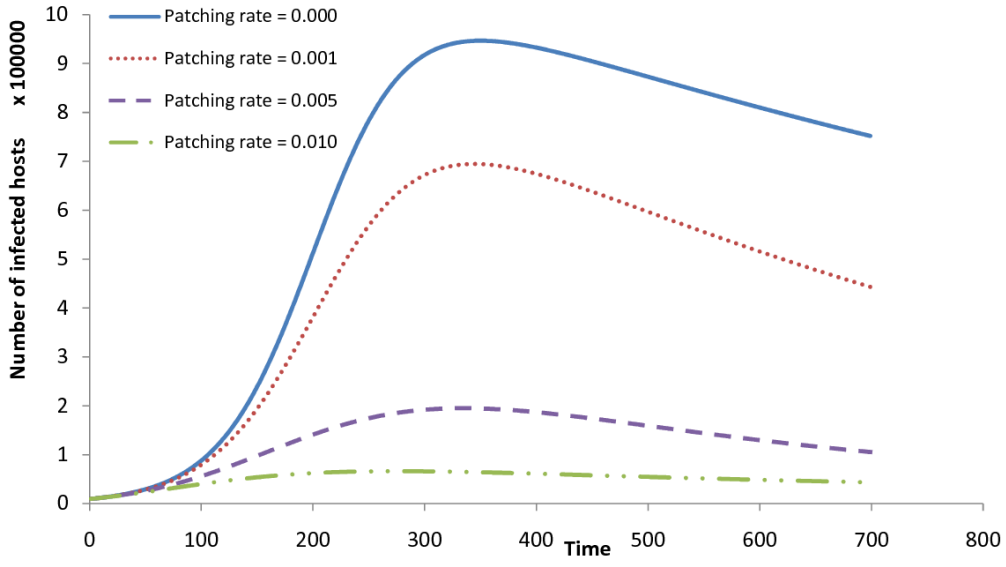

Figure 2.7.: The malware propagation with different patching rates.

2.5.5 The Effect of Different Patching Rates

We performed experiments to evaluate the effects of different patching rates on the malware propagation. In this experiment, we set d to 0.0001, r to 0.0003, k to 100, and the hitlist size to 10,000.

Fig. 2.7 shows the malware propagation with patching rates ranging from 0 to 0.010. We observe that patching significantly reduces the number of infected hosts. Specifically, we observe that the malware propagation peaked at 900,000 hosts when there was no patching, and the number of hosts dropped to approximately 700,000 when the patching rate was just 0.001, which means that only one out of one thousand hosts is patched. The malware propagation peaked at only 193,000 hosts when the patching rate was 0.005, and the malware propagation was significantly reduced and peaked at only 66,000 hosts when the patching rate was 0.01 (one out of one hundred hosts). Therefore, we conclude that patching can significantly diminish the malware propagation and should be employed in all networks.

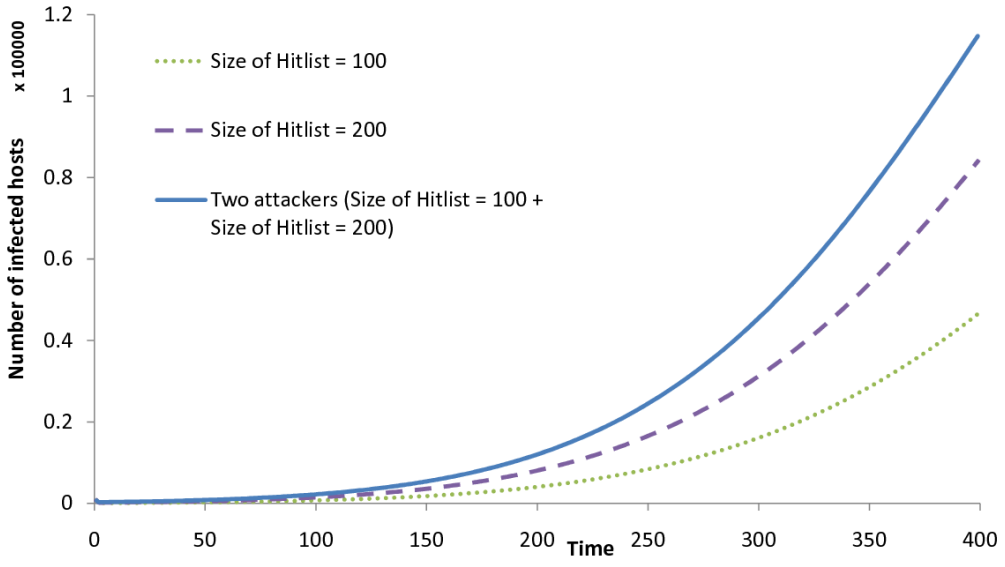

Figure 2.8.: The malware propagation with multiple attackers.

2.5.6 The Effect of Multiple Attackers

We conducted experiments to study the effects of multiple attackers on the malware propagation. In this experiment, we set d to 0.005, p to 0.005, r to 0.03, and k to 100. We first set the hitlist size to 100 and 200, respectively, and performed the experiments. Then, we simulated the multiple-attack scenario discussed in Case 2 of Section 2.4.3: there are two collaborative attackers, one with hitlist size 100, and the other with hitlist size 200. The two attackers start at the same time, and communicate with each other to avoid duplicate infection attempts.

According to Equation 2.33, the effects of a collaborative attack is the sum of the individual attacks. Fig. 2.8 presents the experimental results. The dotted line represents the propagation with hitlist size 100. The dashed line represents the propagation with hitlist size 200. The solid line represents the propagation with two attackers, one with hitlist size 100 and the other with hitlist size 200. The number of infected hosts for the collaborative attack is approximately the sum of the number of hosts infected for the individual attacks with hitlist sizes 100 and 200 initially, which confirms Equation 2.33. However, we note that after around time tick 300, the sum

of the number of infected hosts for the individual attacks become larger than the number of infected hosts for the collaborative attacks, which means:

$$ITOTAL_{x_{total},y_{total},j} < I_{\sum_{n=0}^{m-1} x_n,\sum_{n=0}^{m-1} y_n,j}$$

The explanation is that the number of infected hosts for the collaborative attacks increases more slowly due to the contention between the collaborating attackers. In Case 2 of Section 2.4.3, we assume that the collaborative attackers can avoid duplicate infection attempts, but cannot avoid duplicate scanning. Hence, some scanning activities in the collaborative attack may collide and such collision can decrease the efficiency of the collaborative attack.

2.5.7 The Effect of Different Propagation Times

We performed experiments to evaluate the effects of different propagation times on the malware propagation. In this experiment, we set d to 0.005, p to 0.005, r to 0.003, k to 100, and the hitlist size to 1,000. Note that these destruction and patching rates are high. The sum of the two rates is $0.005 + 0.005 = 0.01$. Note that the birth rate is 0.003, which is smaller than the patching rate.

Fig. 2.9 shows the malware propagation with different propagation times: 2 time slices, 20 time slices, and 50 time slices. We observe that as the propagation time increases, the propagation speed decreases. In 200 time slices, the malware infected 38,416, 13,249, and 5,512 hosts, with the 2-time-slice, the 20-time-slice, and the 50-time-slice propagation times, respectively. In 300 time slices, the malware infected 125,980, 39,497, and 13,403 hosts, with the 2-time-slice, the 20-time-slice, and the 50-time-slice propagation times, respectively. Furthermore, the malware propagation reached its peak at time tick 482 with 263,732 infected hosts with the 2-time-slice propagation time, while the malware propagation with the 20-time-slice and the 50-time-slice propagation times are still in the process of trying to infect more nodes. Therefore, we conclude that the defenders fighting malware should try to maximize its propagation time.

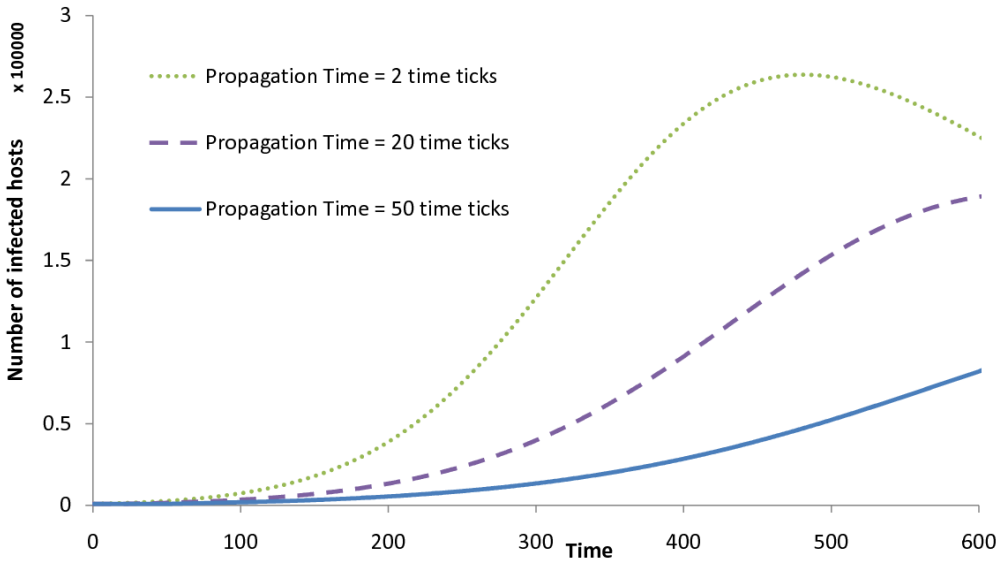

Figure 2.9.: The malware propagation with different propagation times.

2.5.8 Comparison With Existing Models

In this subsection, we show that our model is better by comparing it to existing models. Fig. 1 in [18] shows the results obtained by the AAWP model by Chen et al.. Their experiments employ one million vulnerable machines, a scanning rate of 100 scans/second, a death rate of 0.001/second, and random scanning.

The leftmost graph in Fig. 1 in [18] illustrates the effects of hitlist size. In comparison, our results (Fig. 2.2, 2.3, and 2.4) verify the important shift property, and measure the impact of multi-threading and hitlist size on malware propagation. Our results are more practical since threading is employed by most real-world malware. The middle graph in Fig. 1 in [18] illustrates the effects of patching rate. In comparison, our results (Fig. 2.6 and 2.7) provide more insights into the effects of patching/birth/death rates. We incorporate not only patching and death rates in our studies, but also the birth rate. Our results on the birth rate (Fig. 2.6) are significant, especially for wireless and peer-to-peer networks, in which hosts may join or leave at any time. Our results show that death rate can cause the number of infected hosts to decrease over time, which cannot be inferred easily from Fig. 1 in [18].

The rightmost graph in Fig. 1 in [18] illustrates the impact of infection time. While their results show that infection times do not affect malware propagation significantly, our results show otherwise. Our explanation is that we consider the impact of threading, infection time, and multiple-attacker collaboration, as well as the effects of hitlist size, birth rate, and patching rate on malware propagation. Our results are more intuitive because modern malware has very high propagation speed and longer infection time leaves less vulnerable hosts infected.

In all, the comparison shows that our model is more accurate and complete by considering the issues of MMIMC, and provides more insights into malware propagation.

2.6 Conclusion

We quantitatively study issues of Multi-port scanning, Multi-threading, Infection time, Multiple starting points, and Collaboration (MMIMC) in malware propagation. To our knowledge, there is no previous study on the effects of MMIMC. We discuss the limitations of current models, and explain the impact of threading, infection time, and collaboration, as well as the effects of hitlist size, birth rate, and patching rate. We consider the multi-threading issue during the calculation of probability of successful scans. We model the infection time and propagation time of the malware by employing Fibonacci Number Sequence. We derive the Shift Property and the Linear Property. We theoretically analyze the effects of the above issues, and perform experiments to verify the theoretical results.

3 ALLOCATION SCHEMES, ARCHITECTURES, AND POLICIES FOR COLLABORATIVE PORT SCANNING ATTACKS

3.1 Introduction

Attackers employ various technologies to launch attacks, such as Denial-of-Service (DoS), BotNet, Worm, and Virus, *etc.* The first step of these attacks is to discover vulnerable victim hosts.

Nearly all attackers perform port scanning to find vulnerabilities on victim hosts. Most existing fast-replicated viruses and worms [28], [8], [9] perform port scanning to discover and infect targets. Hence, it is crucial to study port scanning and explore whether the latest advances in technologies have changed the horizon of port scanning, including how to perform port scanning, expedite scanning speed, conduct port scanning from multiple machines, and defend against modern port scanners.

Different network protocols employ different ports. Vulnerabilities exist in all protocols. Hence, to gather information completely, port scanners have to perform scanning for a large number of ports. The size of the port space is 65535 [36]. Ports 0 to 1023 are well-known ports, ports 1024 to 49151 are registered ports, and ports 49152 to 65535 are dynamic or private ports.

Port scanners must run extremely fast. Port scanners have employed sophisticated techniques to expedite port scanning. For example, worms can search vulnerabilities on a commonly used port (e.g., port 21 used by FTP, and port 443 used by HTTPS). However, a typical complete port scan is time-consuming. For example, a 65,536-port UDP scan for one target host could take more than 18 hours [19].

Attackers typically perform port scanning independently, without coordination, to find victim hosts. If port scanning software packages are run on multiple machines without coordination, their search spaces will overlap significantly. The overlap causes

reduction in the performance of the scanning. The network connections used by the port scanners could get congested. The buffer size of the network software may not be large enough to hold all the incoming data. The processing speed of the computer may not be enough to analyze responses from all the networks. After all scanning activities end, all the computers involved in the scanning must communicate to each other and finalize the search results. Problems arise when their results differ. Such differences are hard to analyze, due to the fast-changing nature of computer networks.

As defense technologies evolve, port scanners that exhibit unusual network behaviors, such as sending requests to all IP addresses in a Class B network, are more likely to be detected. Such detection will likely disable the machine performing the scanning immediately and trigger chained detections of all other machines involved. Given the fact that virtually all networks are protected by firewalls, filters, and monitors, a simple deployment of identical port scanning software packages to all computers involved in the scanning is not acceptable.

A key observation to the above deployment plan is that there is a lack of collaboration among the port scanners. We use collaboration and coordination interchangeably. A simple increase in the number of port scanners creates too much duplicate work, increases the power of the whole attack incrementally, and introduces overhead on analyzing, comparing, and resolving the conflicts in the results. Therefore, we need a smarter deployment plan which makes full use of all scanners involved, avoids performing duplicate actions to the maximum extent, and synchronize actions of participating nodes properly and efficiently.

We propose a smart and efficient deployment plan of port scanner software packages to multiple computers. To address the issues above, our deployment plan employs the Distributed Hash Table (DHT) to speed up the scanning, avoid duplicate scanning and contention, and efficiently process and summarize the results from multiple computers. The key idea is to perform DHT lookups on the target host before initiating any scanning activity. In general, the idea can be extended to other collaborative attacks as well.

DHTs [58] are distributed systems that provide essential functionalities of hash tables (HT). In HTs, one can insert items and query whether a particular item exists. For each stored item, an associated value can be retrieved in the HT. Similarly, in DHTs, one can insert items and query for the existences of items. (Key, Value) pairs can be stored in DHTs as well.

The difference between a DHT and a conventional HT is that the stored items are distributed over multiple computers. The key advantages of the DHT include:

1. *Efficiency:* DHT is designed to store information and perform lookups efficiently.

2. *Scalability:* DHT is designed to scale to thousands of computers.

3. *Robustness:* Most DHTs can let participating computers to join or leave at any time, and gracefully handle computer failures.

4. *Distribution:* Items stored in DHTs are distributed over a large number of computers. No central server is needed to answer queries.

Numerous application have employed DHT in the past, such as BitTorrent and Emule [54], [55].

DHT fits the collaborative port scanners because it allows efficient lookups of IP addresses (or IP address and port number pairs). In our deployment plan, each local port scanner double-checks an IP address in the DHT before the actual scanning, therefore avoids problems on duplicate scanning and contention. The DHT database serves as a result repository. It can store scanned results as the (IP address, scan result) pairs. Although DHT has associated overhead in insertions and lookups, it provides higher efficiency by avoiding duplicate scanning, contention, and unnecessary data analysis. Hence, the DHT-based collaborative port scanning scheme significantly improves over uncoordinated and unsynchronized scanning.

3.2 Related Work

Port Scan.

Port scanners employ various techniques. In a SYN Scan, the scanner produces its own IP packet and sends TCP SYN packets to victim hosts and analyzes the responses. In a UDP scan, the scanner sends UDP packets to victim hosts and checks whether ICMP port unreachable messages are received afterwards.

Port scanning can be performed on multiple ports. Some scanners perform the scanning in two-iterations. These scanners scan with one technique, e.g., SYN scan, first before scanning the un-denied ports with other techniques. For instance, the famous NMAP scanner [19] uses the two-iteration approach when executed with the -SUV option.

Ref. [53] discussed how to detect coordinated port scans. However, the author mainly focuses on the detection and did not provide details of how to coordinate the individual port scans.

Port scanner needs to scan a large number of ports, as discussed in Ref. [36]. Example ports include port 7 for echo, port 21 for FTP, port 22 for SSH, port 23 for Telnet, port 25 for SMTP, port 80 for HTTP, port 79 for finger, port 110 for POP3, port 139 for NetBIOS, port 143 for IMAP, port 443 for HTTPS, and port 53 for DNS.

Coordination.

Port scanners can collaborate with each other and perform much more efficient reconnaissance. Staniford *et al.* [24] discussed the Warhol worm, which propagates extremely fast by self-coordination with both hit-list and permutation scanning. Wiley [41] described an abstract distributed and collaborative worm Curris Yellow. Gates [53] discussed possible collaborations in port scans.

Wang *et al.* [50] described an advanced peer-to-peer Botnet. The distributed.net [56] used distributed computing to break ciphers. Our work discusses specific issues of the DHT-based scheme, proposes different allocation strategies, and illustrates the scanning architectures and policies.

IPv6 Scan.

Yang [35] discussed how to defend worms in IPv6 networks. Bellovin *et al.* [49] presented worm propagation strategies for IPv6 networks. Kamra *et al.* [51] proposed a DNS-scan method that can achieve high spread rates in IPv6 networks.

Worm Scanning.

Ref. [27] discusses characteristics of worms, including protocol, amount of payload and scanning strategy, etc. Ref. [42] talks about the performance and models of worm propagations. The authors talk about the local preference scans. If there are multiple attackers starting from multiple sources, local preference scans will be much more powerful. Ref. [25] describes a class of worms that target network systems such as routers. Ref. [17] discusses how to minimize the number of scans required to infect hosts. Zhang *et al.* [1] discussed a Fibonacci model of worm propagation.

Defense.

Wu *et al.* [43] proposed a worm detection architecture for various worm scanning techniques. Twycross *et al.* [44] built a virus throttle program that can detect the port scanner based on their abnormal network behaviors. Jung *et al.* [47] developed the Threshold Random Walk (TRW) algorithm to identify malicious remote hosts. Kumar *et al.* [52] presented the analysis of the Witty Worm and inferred about the IP address where the Witty Worm was released. Staniford *et al.* [48] described Spice, a port scanner that can detect stealthy scans.

3.3 Issues on Port Scanning

Most attacks include the *reconnaissance* step, in which attackers explore and discover victim hosts for vulnerabilities and important information for launching attacks, including operating system and firewall status. Port scanners are regularly used to perform such activities.

3.3.1 Conventional Port Scanners

In a *port scan*, attackers scan a number of listening ports on the victim host. This method guarantees that all known vulnerabilities to attackers on the victim host can be discovered, i.e., there is no false negative. However, an exhaustive search is time-consuming. On the other hand, in a *port sweep*, attackers scan a particular port on the a large number of victim hosts. Port sweep can reduce the size of search space significantly, but could ignore vulnerabilities on un-searched ports. It is clear that the optimal strategy is to scan only the common or vulnerable ports. Such strategy is difficult to achieve in practice. It is not uncommon to find attackers that employ a combination of both methods. Note that in a *partial scan*, attackers can only scan the ports that match the vulnerabilities that the attackers want to take advantage of. E.g., if the attacker could launch FTP and HTTPS attacks, ports 21 and 443 could be the only ports that the attacker scans. If there are multiple collaborative attackers, one attacker can provide vulnerability database such as the FTP and HTTPS ports mentioned above, another attacker can perform optimal scanning according to the available vulnerability database.

For each port scan attempt, the result can be:

1. *listening*: the scanned port is actively listening. E.g., provided that the victim host tries to accept the TCP connection request, a successful TCP SYN scan receives a SYN-ACK packet from it.

2. *not listening*: the scanned port is not listening. E.g., if the victim host does not listen on a particular UDP port, a UDP scan directed to that port receives an ICMP port unreachable packet.

3. *unknown*: there is no response from the victim host. The IP packets between the scanner and the victim host may be lost on the way, filtered by firewall, or blocked by the anti-virus software.

Successful scans can yield promising results, including the operating system of the target host, the protocol suite in use, and the open ports, etc. Note in case 3), victim hosts may not trust the machine running the port scanner for a number of reasons, including IP address not recognized, host not residing on the same LAN, etc. In such cases, a collaborative scan launched from hosts that are trusted by victim hosts may be successful. The information gathered by the trusted hosts can be passed to other malicious computers.

3.3.2 Detection of Port Scanners

Security monitors that could detect port scanning activities normally employ simple rules to label potential port scanning activities. E.g., the monitors can check whether there are a large number of probes (denote it as m) within a limited time period (denote it as n seconds) from a particular machine. Note n is normally set to a small number to reduce the burden of the Intrusion Detection Systems (IDS) due to their limited ability to log and analyze network traffic. Researchers have proposed new techniques for detecting port scanning activities, such as advanced techniques that employ machine learning and probabilistic packet inspection [47]. Port scanners that choose scan targets randomly are more likely to be detected because of the large-scale network-indicators generated [39].

3.3.3 Collaborative Port Scanners

Individual port scanners have limited power because they usually employ a specific technology. Ironically, they are more likely to get detected because they scan all targets individually and generate excessive network traffic. Collaborative port scanners can perform the work together. E.g., they can vary the port scanning technologies, the port scanning locations, the methods to choose scan targets, and the ways to divide work among themselves. We discuss possible scanning technologies and methods below.

C1) Port Scanning Technologies

Collaborative port scanners can choose from a variety of port scanning technologies, including but not limited to [45], [47], [48]:

1. *Connect scan*: The port scanner employs system call connect() to scan target hosts. This scan does not require special privileges. After a TCP connection is established to the victim host, the port scanner sends a RST packet to close the connection. One drawback of this scan is that the established connections are logged and easily noticed by security monitors and software packages.

2. *Application scan*: The port scanner employs particular application-layer protocols and sends requests according to the protocol-specifics. Example protocols include HTTP, FTP, and DNS. If the victim host responds to such application-layer requests, the port scanner classifies the corresponding ports as active.

3. *Ident scan*: The port scanner connects to the victim host, and uses a vulnerability in the ident protocol to retrieve usernames on the victim host. E.g., if the port scanner connects to a HTTP server, the ident protocol can be used to look up the username running it.

4. *SYN scan*: The port scanner crafts its own TCP packet. The scanner first sends a SYN packet to the victim host. The victim host responds with a SYN/ACK packet. The scanner records this response and classifies the scanned port as listening and accepting incoming connections. The scanner could then send a reset (RST) packet to end the scan. Since the SYN scan does not establish a full TCP connection, the victim hosts will not run out of buffer space for accepting incoming connections.

5. *UDP scan*: The port scanner crafts its own UDP packet. The scanner sends an arbitrary UDP packet to the victim host. If an ICMP port unreachable message is received afterwards, the scanner knows that the port is not active. However,

reply packets might be dropped on the network route. Certain network security monitors and anti-virus software packages may filter out the UDP packets. Therefore, this scanning technology can be unprecise.

6. *ACK scan*: The port scanner crafts its own probe packet and set the ACK flag. The scanner sends the probe packet to the victim host. If an ICMP unreachable message is received or there is no reply, the port scanner confirms that the probe packet is filtered by firewall or security software packages. Therefore, ACK scan is used to detect whether a particular network link is guarded by firewall or security softwares. If the network link is unfiltered, the victim host will return an RST packet.

7. *FIN and Null scan*: The port scanner produces surreal scenarios and analyzes the responses from the victim hosts. In a FIN scan, the scanner sends a FIN packet to the victim host. If there is no reply from the victim host, the scanned port on the victim host is classified as open or filtered, because a closed port would send an RST packet. Similarly, in a Null scan, the port scanner produces a TCP packet that does not have any flag, and sends it to victim hosts. A lack of response suggests that the target port is either open or filtered.

8. *Cloaked scan*: In cloaked scans, it is very difficult for defenders to figure out the identity of the scanners. Network devices, firewalls, security software packages, and servers can log potentially malicious activities (e.g., a connection without data transfer) and analyze them to find the scanners. Port scanners can use cloaked scans in such cases. Example cloaked scans include: a) *proxy scan*: in which the victim host sees a proxy machine rather than the attacker; b) *fragmented packet scan*: in which the port scanners send fragmented packets that can be combined together at the destination; and c) *implementation-flaw scan*: in which the port scanners exploit implementation flaws in the victim host to perform scanning. E.g., the old predictable IP ID sequence number bug [57] (now fixed) can be employed to do port scanning.

9. *Multi-cast scan*: The port scanners can send packets to a multicast address in this case. The packet is then directed to a large number of victim hosts. The port scanner can fake the source IP addresses so that responses can be directed to other collaborative attackers.

C2) Target Selection Methods Collaborative port scanners can choose from a variety of target selection methods:

1. *Naive scanning*: The port scanner chooses the next IP address to scan according to an uniform distribution. Code red and Slammer worms employed this method.

2. *Local scanning*: The port scanner gives priority to local IP addresses. More specifically, with probability p the port scanner chooses to scan an IP address that shares the same first x bits (x can be any number from 1 - 31) with it, and with probability (1-p) it chooses to scan a random IP address.

3. *Importance scanning*: The port scanner assumes that the vulnerable hosts are unevenly distributed, hence important IP addresses should be scanned first. Ref. [32] proposes the importance-scanning method, assuming that the vulnerable host distributions exist and are obtainable. Ref. [26] proposes the static importance-scanning strategies and assumes that keeping information about uninfected hosts is realistic.

4. *Sequential scanning (nearest neighbor scanning)*: The port scanner chooses to scan the next IP address in the lexicographical order.

5. *Hit-list scanning [24]*: The port scanner employs an existing list of IP addresses and scans those addresses first. Such tables may be easily obtained from multiple sources, such as routing tables and social network profiles. In particular, the hitlist that has all the addresses in the BGP routing table is very easily obtained and effective.

6. *Sampling scanning*: The port scanner can choose to scan the representatives of subnets. After successful scans, the port scanner tries to infect the victim hosts and then start new scanning from the newly infected hosts.

7. *Passive scanning*: The port scanner does not send scanning packets. Instead, it collects and analyzes the network traffic that pass through it.

3.4 DHT-based Collaborative Port Scanners

The fundamental problem for port scanners is to find vulnerabilities on all ports of target hosts, build exhaustive vulnerability database, and prepare for the launch of effective attacks against target hosts. As discussed in Section 3.1, port scanning can start from multiple sources instead of only one. The latter is assumed in current research. Multiple port scanners might perform duplicate scanning or cause contention.

Furthermore, as discussed in Section 3.3.3, port scanners on different machines can employ different technologies to scan the target hosts. It is not only inefficient to let all port scanners try all possible scan methods, but such exhaustive searches are also likely to trigger the alarms of defense systems.

To avoid contention among port scanners and increase scanning speed and power, the collaborative port scanning scheme must define clear work allocation methods for all participating scanners, avoid generating excessive network traffic or leaving traces for tracebacks, and specify when to stop scanning for the scanners. We discuss these issues below.

3.4.1 Static and Dynamic Allocation of Targets

A number of attackers can perform the port scanning simultaneously to make much faster progresses. In this scenario, a number of attackers can divide the work to scan a large number of victim hosts. There are two ways to divide the work:

A1) Static Allocation

The Static Allocation (SA) scheme avoids the duplicate work discussed above. In a SA scheme, the target address space is divided to all collaborative port scanners before the launch of the actual attack. Port number and IP address combinations, scanning technology, and vulnerability checking methods are divisible as well in this scheme. Each port scanner gets its own allocation of the target space, technology (to use), and vulnerability (to check) list. Without loss of generality, we only discuss the allocation methods for the target space.

Collaborative port scanners have a number of ways to define the SA policy to divide the work, i.e., ways to divide the large target address space. Examples include:

1. *Divide the address space by hosts.*

   In this policy, each collaborating port scanner will be responsible for all ports on particular hosts. There are two methods to divide the addresses:

   (a) *random*: This policy divides the target address space randomly to individual port scanners.

   (b) *sequential*: This policy divides the target address space sequentially to individual port scanners. Each scanner will be responsible for a chunk of the huge IP address space.

2. *Divide the address space by port numbers.*

   In this policy, each collaborating port scanner will be responsible for the same port on all hosts. There are two methods to divide the addresses as well : random and sequential.

The SA scheme does not address a number of issues, including node failure and dynamic node joining and leaving. Note that in real attack scenarios, such as the worm scanning and propagation, newly infected hosts might join the existing attackers. Also, not all port scanning activities are equal. For instance, routers and firewalls

only filter packets from particular sources. Hence, port scanning packets from one scanner may be filtered, while the packets from another may go through. Moreover, port scanning can be done in different ways. One can initiate the port scanning with different power, such as packet sending rates. One can have unsuccessful scans due to network data loss and jittering. One can get active defense from defenders of the target systems and could even get detected and physically disabled by them. SA scheme fails to address these issues either.

A2) Dynamic Allocation

The Dynamic Allocation (DA) scheme does not pre-allocate target spaces and allows the attackers to divide the work on the fly. In this scheme, attackers can communicate with each other to dynamically determine the next hosts to scan. A key advantage of communication between attackers is that the scanning space can be constantly updated.

In the state-of-the-art port scanners, e.g., the hit-list based worm scanners, the hit-list is divided by half each time using a top-down allocation approach when a new propagation is successful. In contrast, the DA scheme allows the scanning space to be constantly updated.

In the DA scheme, one can develop distributed lookup tables to query whether a particular IP address has been scanned/infected or not.

A3) Hybrid Allocation

The Hybrid Allocation (HA) scheme combines the SA and DA schemes together. If the number of target hosts is large and the number of available attackers is not small, the system can divide the target hosts statically in the first step. The statically divided hosts are then assigned to different groups of attackers. The attackers for particular groups use the DA scheme to generate scan targets. For instance, to scan all target hosts a country, one can divide them by states, and allocate hosts in different states to different groups of attackers. The HA scheme is most useful when there are lots of target hosts and attackers.

3.4.2 Synchronization of Collaborative Port Scanners

Based on the above discussion, in order to synchronize the actions of collaborative port scanners, we need to develop a dynamic or hybrid allocation scheme that allocates the scanning targets to individual port scanners. The DA or HA scheme must be extremely efficient so that it can respond to multiple requests from a large number of collaborative port scanners. During the on-the-fly target allocation, the DA scheme needs to make sure that two conditions are met:

1. No two port scanners will be scanning the same IP address.
2. A port scanner will not be scanning any IP address that has already been scanned by another port scanner.

To facilitate our discussion, we define the status of an IP address based on whether it has been scanned.

Definition 1. Port Scanning Status (PSS)

The Port Scanning Status (PSS) of an IP address is a two-bit number that indicates its the scanning information. More specifically, the PSS of an IP address i can be one of the following:

1. *00* — denotes that the IP address has never been scanned by any collaborating port scanner;
2. *01* — denotes that the IP address is currently being scanned by a collaborative port scanner;
3. *10* — denotes that the IP address has already been scanned by some port scanner.
4. *11* — denotes that the IP address has already been attacked by some attacker based on the scanning results.

Definition 2. Degree of Collaboration (DC)

The Degree of Collaboration (DC) for a collaborative port scan is an integer that records the number of active collaborative port scanners. Since individual port scanners may join and leave at any time during a collaborative port scan, the number of active collaborative port scanners vary from time to time. Hence, the DC for a collaborative port scan at time tick t is :

DC_t = the number of active port scanners at time t.

Definition 3. Collaboration Architecture (CA)

Collaborative attackers need to communicate to each other over the network to synchronize their actions. In particular, collaborative port scanners need to synchronize their scanning activities on IP addresses. To effectively communicate the PSS of an IP address and keep themselves updated about the DC, they need an efficient and robust distributed query system. The basic functionality of the query system is to store scanned results and provide real-time scanning status. Information like the scanned IP addresses, ports, and vulnerabilities are stored in the system. To facilitate discussion, we consider the case that only IP addresses are stored.

There are a number of possible architectures for this query system :

1. *Flooding architecture*: As shown in Fig. 3.1, in this architecture, each collaborating port scanner "floods", i.e., broadcasts messages to, all known collaborators to query the scanning status of a particular IP address. While robust against node failures, this architecture requires that each node stores its own scanning status information and incurs significant network overhead due to the huge amount of query traffic.

2. *Collaboration-server based architecture*: As shown in Fig. 3.2, in this architecture, each collaborating port scanner registers itself at a collaboration server dedicated to monitoring scanning status, and joins the collaborating port scanner group. If a collaborating port scanner stops the scanning activities, it will notify the collaboration server. The collaboration server is responsible for stor-

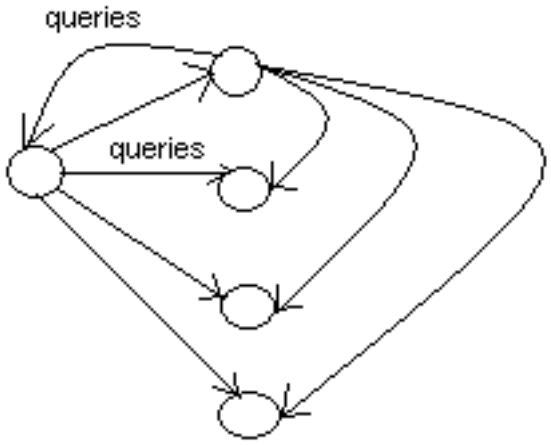

Figure 3.1.: The flooding architecture.

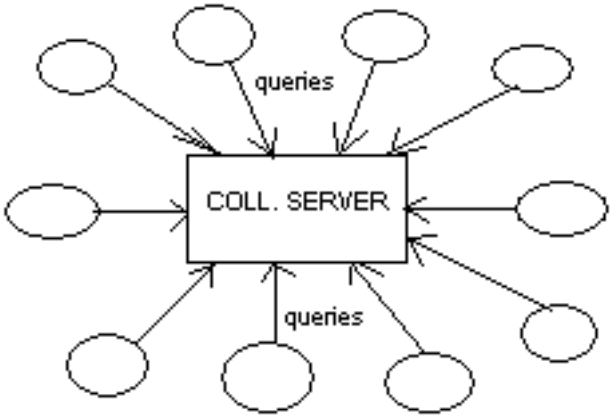

Figure 3.2.: The collaboration-server based architecture.

ing the scan status and results for all port scanners. Each collaborating port scanner queries the collaboration server for real-time scanning status and makes decision on the next scan target. While efficient, the reliability of this architecture depends on that of the collaboration server. If the collaboration server has limited bandwidth, it will not able to handle the large amount of network traffic generated by individual port scanners.

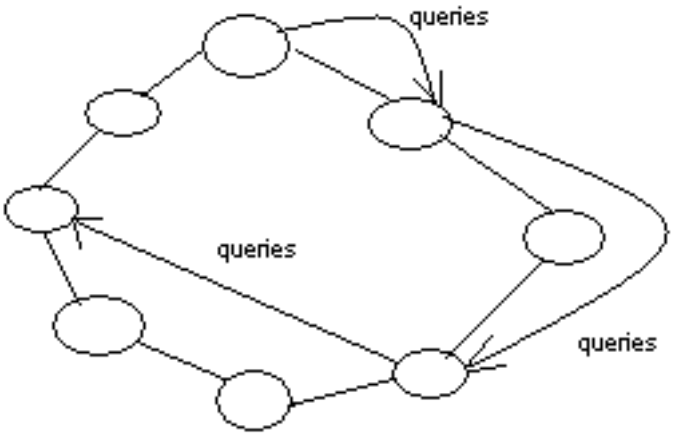

Figure 3.3.: The distributed architecture.

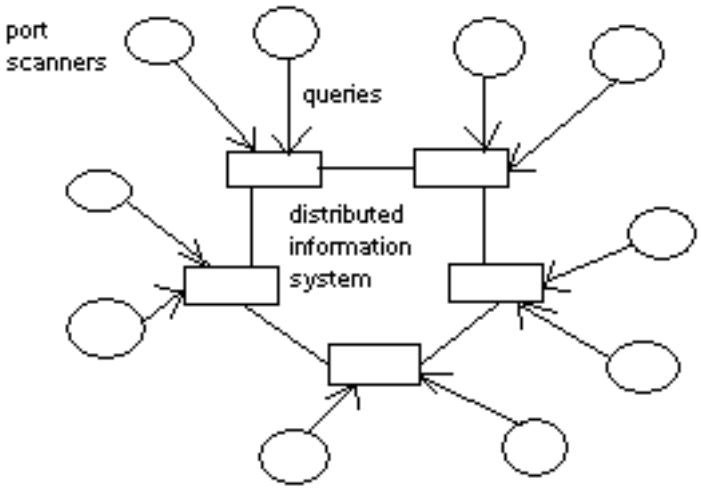

Figure 3.4.: The hybrid architecture.

Moreover, the collaboration server is a single failure point. The collaborative port scan could not proceed if the server is down. Even if a new collaboration server can be established, a lot of efforts and time need to be spent in the recovery. The collaboration-server architecture is vulnerable to defense as well, e.g., the defenders can analyze the traffic patterns of the collaboration server, determine that it is acting as a communication and command center, and take it down to shut down the whole collaborative port scan. In the real world, researchers have proposed traffic analysis methods to defend against Botnets based on IRC channels [11], [40].

3. *Distributed architecture*: As shown in Fig. 3.3, in this architecture, the scanning status of all collaborating port scanners are distributed over all the scanners. Therefore, each collaborating port scanner issues queries to the distributed information system based on themselves. The distributed information system acts as the efficient storage and query server, scales to a large number of nodes, and provides high reliability.

   This architecture eliminates the concentration of information and network traffic on the collaboration server. Its efficiency is much higher than the flooding architecture. However, each port scanner has to both perform scanning and serve as a active node in the distributed information system.

4. *Hybrid collaboration architecture*: As shown in Fig. 3.4, in this architecture, there are two groups of collaborating attackers: the first group of them is the traditional port scanner group, and the other is the information group, i.e., the one responsible for the distributed information system discussed in the distributed architecture. The attackers from the first port scanner group query the attackers from the information group for IP address scan status. The attackers from the information group builds, indexes, and stores all scan status information efficiently. Attackers from the port scanner group view the distributed information system as a collaboration server discussed in 2).

The hybrid collaboration architecture combines the collaboration-server based architecture and the distributed architecture. Compared to the distributed architecture, the hybrid architecture relieves the port scanners from infrastructure issues, i.e., storing scanning status information and answering IP scan status queries. It specifies a dedicated group of attackers responsible solely for the distributed information system on scanning status. Hence, attackers do not have to balance their resources between the actual attacks and the infrastructure.

By such task division and collaboration, attackers take advantage of the benefits of both collaboration-server based and the distributed architectures. Therefore, they are more likely to increase the efficiency and the scalability of their systems and launch much more powerful attacks.

3.4.3 The DHT-based Contention-Avoidance Allocation Scheme

Overview

We need a distributed port scanning system that can avoid duplicate scanning and contention among collaborative port scanners. duplicate scanning and contention include simultaneous scanning of an identical victim host, generation of excessive network traffic on the same network link, and duplicate work of distributed port scanners, i.e., scanning the same port on the same victim host for an identical vulnerability.

The proposed DHT-based collaborative scanning scheme can elegantly avoid duplicate scanning and contention. The scheme incorporates the well-designed distributed lookup system, the DHT, that stores scanning status information and answers queries from collaborative port scanners. The DHT provides distributed look-up services. Based on the discussion in Section 3.4.2, The collaborative port scanning scheme employs the distributed/hybrid architectures because they provide higher efficiency and scalability.

Traditional port scanners send probing packets and analyze responses from victim hosts. In the collaborative port scanning scheme, each collaborative port scanner

queries the DHT before each scanning. Note besides DHT, there are other candidates for the distributed information storage and query system, such as the Big Table [6]. If false positive can be tolerated, Bloom Filter [7] can help with the query system as well.

The DHT-based scanning algorithm

Each collaborative port scanner runs the new DHT-based scanning algorithm in the proposed scheme.

The algorithm for the collaborative port scanner is presented in Algorithm 1. The collaborative port scanner picks up an IP address and a port number, and checks if the IP address and port number combination has been scanned already. If not, the port scanner performs port scanning activities using a randomly chosen scanning technology discussed in Section 3.3.3, and records the scanned results. There are two important methods for the collaborative port scanner: the GET() method, responsible for checking the scan status for IP address and port number combinations, and the SET() method, responsible for recording the scanned results.

The algorithm for the GET() method in the scanner is presented in Algorithm 2. The GET() method processes the (IP address, port number) pair, and looks it up in the DHT to check its scan status. If there is a match, the GET() method returns a variable indicating that the (IP address, port number) pair has already been scanned. Otherwise the GET() method returns a variable indicating that the pair has not been scanned. To perform look-ups in the DHT, the GET() method can employ RPC calling mechanisms. Note that DHT performance optimizations allow fast lookups. E.g., caching and the hybrid architecture discussed above can significantly reduce the lookup latency.

The algorithm for the SET() method in the scanner is presented in Algorithm 3. The SET() method processes the (IP address, port number) pair, and records its status in the DHT. Note the SET() method must take concurrency control issues into

consideration because no concurrent GET() and SET() should be allowed on the same (IP address, port number) pair to ensure correctness of the whole system. Although concurrent GET() accesses are allowed, no two SET() methods should be modifying the scanning status of the same (IP address, port number) at the same time. This problem is similar to the reader/writer lock problem: concurrent GET()s is allowed while concurrent SET()s and concurrent GET()s/SET()s are prohibited. The GET() and SET() methods can implement a reader/writer lock in this case to improve the lookup performance. Another way to improve the performance is to lock only part of the DHT. By default, given an (IP address, port number) pair, the SET() method can request to lock certain IP address ranges instead of the whole IP range. E.g., if the SET() method requests to lock only the Class B network that the given IP address belongs to, other SET() methods can write to other class B network addresses, which improves the performance of the whole system.

3.4.4 Detection Avoidance

An effective way to detect traditional port scanners is to watch for abnormal network traffic patterns. As discussed in Section 3.1, thresholds such as excessive number of pings within a certain time period can be set up to trigger alarms for port scanning activities. An obvious "solution" is to perform "stealth scans", e.g., perform scanning activities slowly for several months and gather the results. Such solution cannot get enough information quickly and is not desirable for the port scanners.

Collaborative port scanners can distribute the work among a large number of machines that are in different geographical areas, thus reduce the network traffic generated by individual port scanners and avoid detection. An optimal scanning strategy for detection avoidance is to "blend into the crowds", i.e., to mix scan traffic into normal network traffic and make it difficult for defenders to notice. E.g., smart scanning schemes that resemble the collaborative port scanners as web crawalers, bots, or spiders could successfully foil a large number of defense systems. Moreover, by employing

Algorithm 1 The Collaborative Port Scanner

```
// Get an unscanned IP address and port number combination.
repeat
  // Do preprocessing works.
  //
  ip = ChooseIPAddressToScan();
  port = RandomlyPickUpAnPortNumber();

  // Check with the DHT to see if the IP address
  // and port number combination has been scanned already.
  // The GET() method returns NOT_SCANNED if the
  // IP address and port number combination has not been
  // scanned yet.
  scan_status = GET(ip, port);
until scan_status = NOT_SCANNED

// Perform the scanning activities. The scanning method
// is generated randomly from the scanning technology
// database, including the ones discussed in Section 3.3.3.
scan_method = RandomlyPickUpScanningMethod();
scan_method.Send(probing packets, victim);
scan_method.Receive(responses, victim);
scan_method.scan_result = Analyze(responses);

// Record the result of the scanning to the DHT.
SET(ip, port, scan_method.scan_result);

// Return.
return OK;
```

Algorithm 2 The GET Method

```
// The GET() method :
// takes:
// (ip address, port number) pair as inputs; and
// returns:
// NOT_SCANNED : if the pair has not been scanned yet;
// SCANNED : if the pair has been scanned already.

Require: IP address and port number are correctly passed in as arguments.

// Do preprocessing works.
ip_port_pair = GeneratePair(ip, port);

// Contact the DHT to read information.
RPCCallBack = SetupRPCCall();
RPCCallBack.Run();
WaitForRPC();

// Get the scanning status for the (IP address, port number)
// pair.
scanning_status = ProcessRPCResults();

// Return.
if scanning_status = 0 then
  return NOT_SCANNED;
else
  return SCANNED;
end if
```

Algorithm 3 The SET Method

```
// The SET() method :
// takes:
// (ip address, port number) pair and
// the scanned result as inputs;
// performs:
// DHT information updates.

Require: IP address, port number, and the scanned result are correctly passed in as
  arguments.
  // Do preprocessing works.
  ip_port_pair = GeneratePair(ip, port);
  // Request exclusive access for the DHT.
  Lock(starting IP address, ending IP address);
  // Contact the DHT to write information.
  RPCCallBack = SetupRPCCall();
  RPCCallBack.Run();
  WaitForRPC();

  // Check if the update was successful.
  CheckSuccess();
Ensure: Update is successful.
  // Release the lock.
  Unlock(starting IP address, ending IP address);
Ensure: lock is released.

  // Return.
  return OK;
```

the distributed and the hybrid architectures discussed in Sec.3.4.2 to distribute network traffic, collaborative port scanners can escape detection of intelligent defenders. Methods that analyze the network traffic using data mining techniques [40] to identify command centers of collaborating malicious computers will fail to locate collaborative port scanners.

3.4.5 Stop Policy

A critical problem for the collaborative port scanners is to determine when to stop the scanning activities. Optimally, the collaborative port scanners stop after all hosts has been scanned for every possible vulnerability. In practice, this mission is difficult to accomplish because each host needs to make its own stop decision based on its knowledge of the global scanning activities. Ref. [37] proposes an autonomous design. In their design, each host employs a Sum-Count-X method to determine when to stop, and communication among hosts is necessary to improve the precision of stop estimation. Ref. [38] proposes a quorum-sensing design. However, they did not consider the network topology. Also, after the stopping of the worm propagation, a worm user needs to manually restart it.

In our approach, The DHT records all scan statuses. We can constantly monitor the uninfected nodes as long as there are empty entries in the collaboration table. Therefore, the stop condition for the collaborative port scanners can be defined as all (IP address, port number) pairs have been scanned, as reflected in the DHT. We believe the collaborative-table approach is faster and more efficient since the DHT serves as the monitor and recorder of all status messages. No approximate calculation is used. If not all hosts or ports need to be scanned, users can relax the definition of the stop policy. E.g., collaborative port scanners can be defined to stop after 90 percent of all hosts are scanned.

3.4.6 Target Selection and Revisit Policy

A key step in the DHT-based collaborative port scanning is to generate random IP addresses and port numbers. Random number generators can be exploited to analyze the bandwidth of worm senders [52]. Hence, one enhancement to the scheme is to employ a variety of random number generating methods. By varying the way to generate random IP addresses and port numbers, the collaborative port scanning becomes polymorphic, and is much more resistant to analysis and defense.

To improve the efficiency of the collaborative port scanners, learning algorithms can be employed. The DHT stores a lot of information, which can be analyzed to improve future scanning activities. Moreover, the collaborative port scanners can scan a portion of all IP addresses and port numbers to reduce the scanning time. For example, they can selectively scan only one IP in a Class C subnet, and use the scanning results to infer information about other target hosts that reside in the same subnet. Some IP addresses are employed by honeynets. To prevent collaborative port scanners from being detected and analyzed by such honeynets, the corresponding entries in the DHT for these addresses can be marked as "do not scan".

Host configurations and vulnerability statuses change over time. To capture the changes, collaborative port scanners should revisit the hosts. Some hosts have dynamic IP addresses. In such cases, collaborative port scanners can periodically revisit them and update the information on the hosts and IP addresses. A simple revisit policy is the Age Policy. In the Age Policy, there is an age attribute for each DHT entry. The age is increased by the DHT automatically and checked against a threshold. If the age reaches the threshold, the system purges its entry from the DHT to initiate a new scan. Note that researchers have studied revisit policies for web crawlers [59] and such policies could be adapted.

The DHT-based scheme needs to handle errors of participating nodes. E.g., participating nodes may provide incorrect scanning results. A simple solution to this problem is to set the revisit policy to allow each target to be scanned twice within

each scanning period. Then the system could compare the results to decide if the results are usable.

3.4.7 Comparisons and Caveats

The DHT-based collaborative port scanning scheme can scan multiple vulnerabilities on multiple ports. In contrast, Botnets and the Curious Yellow worm [41] typically propagate by exploiting a known vulnerability on a certain port. In practice, if the target hosts do not have such vulnerability or have applied patches to fix it, the propagation will fail. Some real-world worms, e.g., the Witty Worm [33] can check for multiple vulnerabilities. In the DHT-based scheme, attackers share knowledge about the progress and information with each other. Therefore, attackers can check for a large number of vulnerabilities and choose one to exploit.

While the extended hit-list methods consist of information on IP addresses, the DHT-based scheme records information on not only IP addresses, but also attributes for each host or subnet. Example attributes include whether the hosts are web, mail, or DNS servers, a ranked list of existing vulnerabilities, and the name and version of the host operating system. During the collaborative port scanning information regarding the configuration of victim hosts can be fingerprinted to facilitate the launch of future attacks.

3.5 Experiments

In this section, we present the experimental evaluation of the DHT-based collaborative port scanning scheme. We conduct experiments to verify our theoretical analysis, in particular: the impact of the DHT-based scanning scheme.

3.5.1 Experiment Setup

We have conducted the experiments by employing OpenDHT [83] on PlanetLab. OpenDHT runs on a large number (around 200) of PlanetLab nodes. Fig. 3.5 shows the architecture of our experimental DHT platform. We employ the hybrid architecture discussed in Section 3.4.2. Queries issued from collaborative scanners can be forwarded to the OpenDHT. Note that we utilize this architecture to learn about the latency of DHT.

The experimental network consists of Intel Dual Core workstations and virtual machines running Windows XP. Without loss of generality, we use IPv4 networks and set the size of target IP address space to 2^{32}. The scanning methods used in the experiment include TCP scanning, UDP scanning, and version detection [19] that could return the system and version information running on the target hosts. We ran port scans on the target hosts to understand the latencies associated with scanning. Such latency information are used to simulate port scanning. Our experiments then simulate the actual port scanning attacks and the DHT latencies.

3.5.2 Experiments on the Performance of the DHT-based Collaborative Scanning Scheme

We conduct experiments to study the performance of the proposed collaborative port scanning scheme. In our experiment, there are 1,000 target hosts that need to be scanned. We compare the performance of 4 different scanning setups:

1. 10 collaborative port scanners. In this setup, there are 10 collaborative port scanners that employ the fast intelligent DHT-based collaborative scanning scheme. The 10 collaborative port scanners divide the 1,000 target hosts into 2 groups. Each group has 500 target hosts. They conduct the scanning group by group, i.e., they only start to scan the second group of targets after finishing scanning the first group of targets. (We discuss more on the collaboration methods in the next experiment.) The collaborative scanners keep each other in-

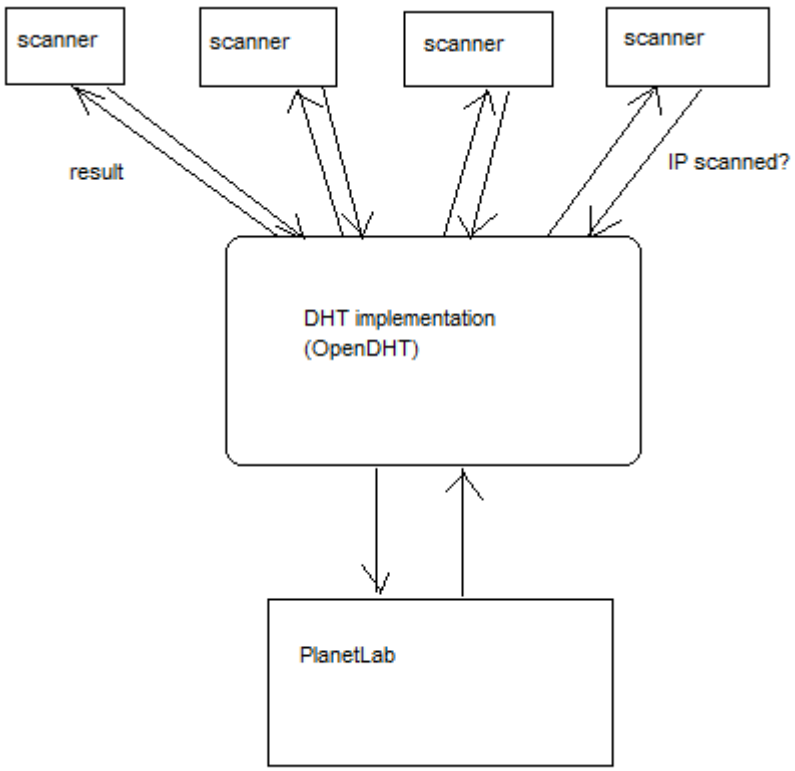

Figure 3.5.: The network topology of the OpenDHT lookup.

formed of the progress of the whole scanning through the DHT. The algorithms employed by the 10 collaborative port scanners are discussed in Section 3.4.

2. 10 port scanners that operate with the static division scheme. In this setup, the 10 port scanners are divided into 2 groups, and each group has 5 port scanners. The two groups of scanners operate individually. Within a group, the 5 port scanners divide the targets statically into 5 parts, and each of them will be responsible for approximately one fifth of the target hosts.

3. 10 port scanners that operate individually. In this setup, there are 10 port scanners, but they just scan randomly without communicating with each other. As soon as the port scanner finishes scanning 50 hosts (one twentieth of all target hosts), it reports the results to a central node, and waits for the signal from the central node. The central node collects the scanned results from all scanner nodes, and combines their results. As soon as the central node finishes combining scanning results from all 10 scanners, it sends signals to all 10 port scanner nodes. In the next round, each port scanner only scans targets that have not been scanned in previous rounds. This procedure is repeated until all hosts are scanned.

4. a single port scanner. In this setup, there is only one port scanner. It is responsible for scanning all the target hosts.

All port scanners scan the well known ports for each victim host in this experiment. We impose a limit of 20 on the number of connections which a single port scanner can initiate to a target host. A typical DHT lookup takes approximately 3 seconds with 10 active scanners. Note that a scanner has to write the results to DHT after a successful scan. A typical port scanning for one victim host that covers TCP ports and version information takes 2 minutes.

Fig. 3.6 shows the number of successfully scanned hosts over time for the 4 scanning setups.

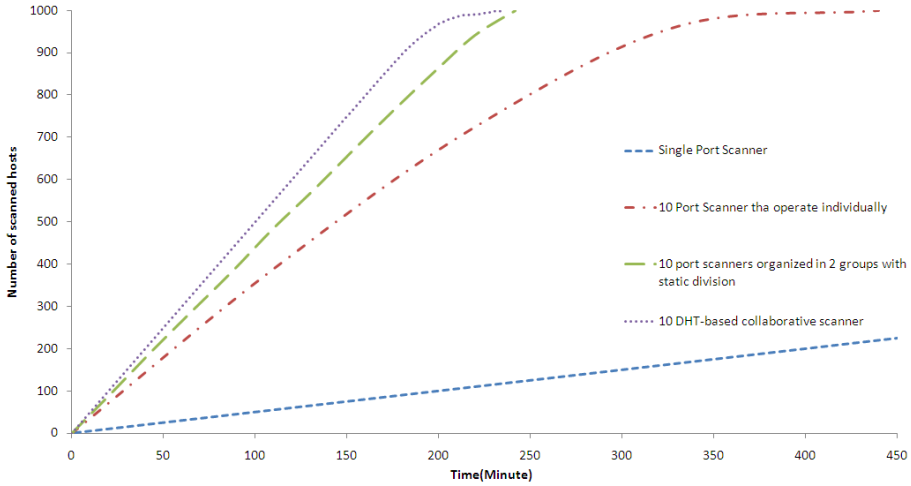

Figure 3.6.: The performance of the DHT-based collaborative scanning scheme.

We observe that, for the DHT-based 10 collaborative port scanner setup, on average, the time of the work spent in the core port scanning part constitutes most of the total operation time. It takes the collaborative port scanners 216 minutes to scan all the target hosts. The overhead ratio of the collaboration, including storing and retrieving the scanning status for the target hosts, is approximately 8 percent.

Our results show that the DHT-based 10 collaborative port scanners clearly outperform the other 3 non-DHT-based setups, and that the performance of the single port scanner is the worst among all setups. Our results show that the time to finish scanning all target hosts of the DHT-based collaborative port scanners is approximately 60 percent of the port scanners with static divisions, and 41.7 percent of the port scanners that operate individually. The explanation is that the DHT-based collaborative port scanners are able to perform port scanning concurrently with much more resources and minimal contention. The other 2 setups that employ static division or operate individually could not eliminate duplicate scanning and scan efficiently.

The experimental results verify our analysis and confirm the performance of DHT-based collaborative port scanners.

3.5.3 Experiment on the Number of Participating Collaborative Scanners

In our first experiment, 10 DHT-based collaborative port scanners collaborate with each other to conduct port scanning. Questions then arise as how would the performance of the collaborative port scanners change, as the number of participating nodes change. One would expect that a larger number of participating nodes increase the number of scanned target hosts within a specific time. However, more scanner nodes could generate more network traffic and impose larger overhead on the DHT due to a large number of scanning status lookup and store requests. In the extreme case, an infinite number of participating nodes generate excessive traffic and overburden the

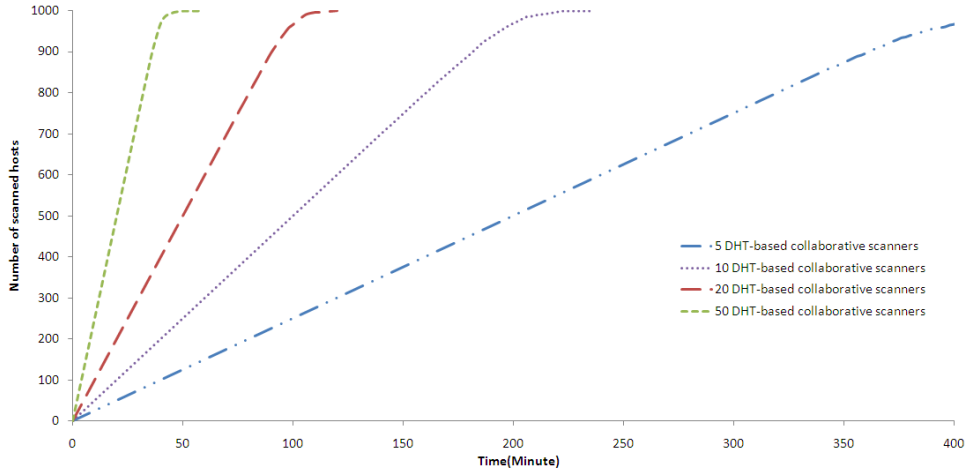

Figure 3.7.: The performance of collaborative scanners with different participants.

DHT, effectively rendering a Distributed Denial-of-Service (DDoS) attack. Therefore, we cannot arbitrarily increase the number of scanner nodes.

We conduct experiments to find out the relation between the collaborative port scanners and the number of participating scanner nodes. Note that different DHT systems and scanning methods have different latencies and could affect such relation. Hence, in our experiments, we vary the scanning methods and use different gateways of the DHT system to create different latencies.

More thorough scans and farther gateways typically have higher latencies. If scanning latency is extremely low when compared to DHT latency, a small number of participating scanner nodes that employ the static division scheme would perform well. The reason is that a large number of nodes overburden the DHT, increase DHT latency, and slow down the whole port scan. When scanning latency is extremely high when compared to DHT latency, a large number of participating scanner nodes perform better. The explanation is that a smaller number of nodes neither have enough parallelism nor fully utilize the DHT system. The most interesting scenario is when neither scanning nor DHT latency is too high or too low . In such cases, a large number of participating scanner nodes incur overhead on the DHT system, but could also overcome the slowness of the scanning itself.

Fig. 3.7 shows the number of successfully scanned hosts over time for different number of participating scanner nodes. We observe that the performance of DHT-based collaborative port scanners increase as the number of participating scanner nodes increases. However, the efficiency of the DHT-based collaborative port scanner scheme decreases as the number of participating scanner nodes increases, which confirms our analysis above.

If the number of participating scanner nodes is approximately the ratio of scanning latency to DHT latency, a good balance between the costs for maintaining a large number of scanner nodes and the performance of the collaborative port scanning can be struck.

Our theoretical analysis is as follows. Denote the number of target hosts as M, the scanning latency as S, the average DHT latency as D, the actual DHT latency as d, the number of collaborative port scanners as n, the ratio of the scanning latency to the DHT latency as k, we have:

$$k = \frac{S}{D}$$

The actual DHT latency increases as the number of collaborative port scanners increases. We assume that the increase is linear, hence:

$$d = D * n$$

The total latency L for one parallel scan is the sum of the scanning latency and the DHT latency :

$$L = S + d$$

The number of parallel scans needed for M hosts P is $\frac{M}{n}$.

Hence, the total scanning time

$$\begin{aligned} T = P * L = (S + D * n) * \frac{M}{n} = (D * k + D * n) * \frac{M}{n} \\ = M * D * (1 + \frac{k}{n}) \end{aligned} \tag{3.1}$$

Note that n should be no more than M. When n = M, the total scanning time reaches its minimum at M * D.

In the real world, we may not be able to include M scanners. However, when n is equal to k, the total scanning time is simplified to 2 * M * D, which is at least half as fast as the fastest possible scanning.

In our experiments, the scanning and DHT latencies are 2 minutes and 6 seconds (on average with 10 collaborative port scanners), respectively. Hence, the optimal number of scanner nodes is 120 / 6 = 20. The experimental results confirm our analysis. With 10 scanner nodes, the overhead of the DHT-based collaborative scanning scheme is approximately 8 percent. With 20 scanners nodes, the overhead is approximately 10 percent. With 50 scanner nodes, the overhead is approximately 30 percent. Note that the quality of the DHT system affects the overhead with respect to different number of scanner nodes. In the real world, if more efficient systems can be utilized, the number of collaborative port scanners can be very large and still does not incur too much overhead.

3.5.4 Discussions on Deployment and Defense

In our experiments, we have examined the performance of the DHT-based scanning system. Large-scale deployment of the DHT-based collaborative attack scheme in the real world needs attention on a number of issues. Defenders of collaborative port scanners can mitigate the attacks by employing countermeasures to these issues.

Detection Avoidance. Certain routers and firewalls check network traffic against pre-set thresholds. Defense mechanisms such as the communication pattern analysis [40] could reveal the centralized servers. The DHT-based collaborative attack scheme could employ several methods to avoid detection: 1) limit the rate for sending and receiving packets; 2) create de-centralized traffic that obfuscate the communication pattern analysis; and 3) employ encryption for the DHT-related packets whenever possible, since most existing defense systems cannot analyze encrypted traffic.

Integrity of the DHT. DHT functionality is crucial in the collaborative port scanning scheme. If the defenders can locate the DHT servers, they could try to disconnect, mislead, and defend against them by offense [20]. Hence, protecting the DHT against misuse, errors, and attacks is very important for the real-world collaborative scanners. Mechanisms that can help protect the DHT include user authentication and encryption of the stored IP address, port number, and scanning status values. User authentication may introduce extra communication latency and hurt the performance of the DHT-based collaborative scanning scheme, while the encryption of the information stored in DHT may increase the time to store and look up scanning statuses.

Scale of the DHT. In practice, the collaborative port scanning system may implement its own fast DHT system. We have seen similar large-scale system in industry, including BigTable [6] of Google and Dynamo [12] of Amazon. The DHT system can run on a large number of attacker nodes in the hybrid mode, e.g., 10,000, in practice. If more attacker machines are allocated to the DHT system, however, there will be fewer hosts responsible for the actual scanning, which can result in deteriorated scanning performance. If the collaborative port scanning system can utilize some publicly-available infrastructure and leverage its power, the overall scanning performance could improve significantly.

3.6 Conclusion

We study the collaborative port scanning, in which attackers collaborate to search the network for open ports that could be exposed to attacks. We propose different collaborative scanning strategies and analyze their advantages and disadvantages. We discuss the static, dynamic, and hybrid allocation schemes and how to employ DHT in the system. We conduct experiments to evaluate the performance and overhead of the collaborative port scanning system.

Our results show that DHT-based collaborative port scanning is a promising approach. It provides good performance, and proves that attacks can be launched by collaboration. As network speed increases in the future, we may witness an increased number of sophisticated collaborative attacks that orchestrate the computing power of a large number of attackers. Our results suggest that issues like the number of collaborative attackers in the system, different methods of collaboration, scanning granularity, and revisit policy all significantly affect the performance of the collaborative port scanners. We discuss issues that need consideration in real-world deployment and defense mechanisms that could mitigate the collaborative port scanners.

4 EXPERIMENTS ON DEFENSE

In previous chapters, we discussed the methods and impact of collaborative attacks, with a focus on port scanning attacks. In this chapter, we develop and experiment with various methods to defend against such attacks.

There are a lot of ideas for defense. E.g., one can use uncommon port numbers, disable not-running services, and use TCPwrapper (Unix tool that performs access control based on IP/domain, etc.) to defend against collaborative port scanning attacks. Our methods include:

1. Delay and vary the response latency for attackers. By slowing down the port scanning attack, we hope that the whole attack will be slowed down. For instance, sending a slow TCP ACK reply would normally foil fast-paced port scanning attacks.

2. Reload the system with different settings and parameters. This includes patching, refreshing, and revisiting (for attackers). Attackers then would have only old information regarding the attacks on file. Therefore they could not launch attacks based on such information. To obtain updated information, attackers have to update their database and re-scan/revisit the victim hosts again.

4.1 Experiments on the Delaying and Varying the Response Latency

To defend against port scanning attacks, defenders can slow down the response to attackers. Meanwhile, collaborative attackers could perform the port scanning with a variety of granularity. In our experiments, response slowdown is modeled as a higher granularity of scanning. The reasoning behind this method is that higher-granularity

port scanning attacks have higher latencies, which resembles the slow response to attackers.

Different granularity of scanning result in different latencies. In this experiment, we compare the performance of different scanning granularity, including scanning 511, 1,023, and 2,047 ports. There are 10 collaborative attackers and 1,000 target hosts in this experiment. We compare its performance against a single port scanner.

Fig. 4.1 shows the performance of the single and the 10 collaborative attackers with different scanning granularity.

We observe that with finer scanning granularity, i.e., more ports are scanned for each target host, the performance of the whole port scanning system decreases. The explanation is that more time are spent in the scanning part and less hosts can be scanned within a given time.

We observe that the performance of the collaborative attackers decreases more slowly than that of the single port scanner. The explanation is that when scanning granularity is coarse, scanning takes a short time, and the collaborative attackers spend significant amount of time in the collaboration, including infrastructure reads and writes. However, when scanning granularity is fine, scanning takes a much longer time, and the collaborative attackers spend a smaller portion of time in collaboration, resulting higher efficiency.

The experimental results also confirm our analysis for the previous experiment. According to Equation (3.1), finer scanning granularity, which results in longer scanning latency, increases the number of collaborative attackers needed to reach optimal efficiency. We can infer from the results that the collaborative attackers can realize its potential in power with complete and large-scale scans, which are much slower. Note it is easier for defense systems to detect complete scans that cover a large number of ports or perform target system fingerprinting.

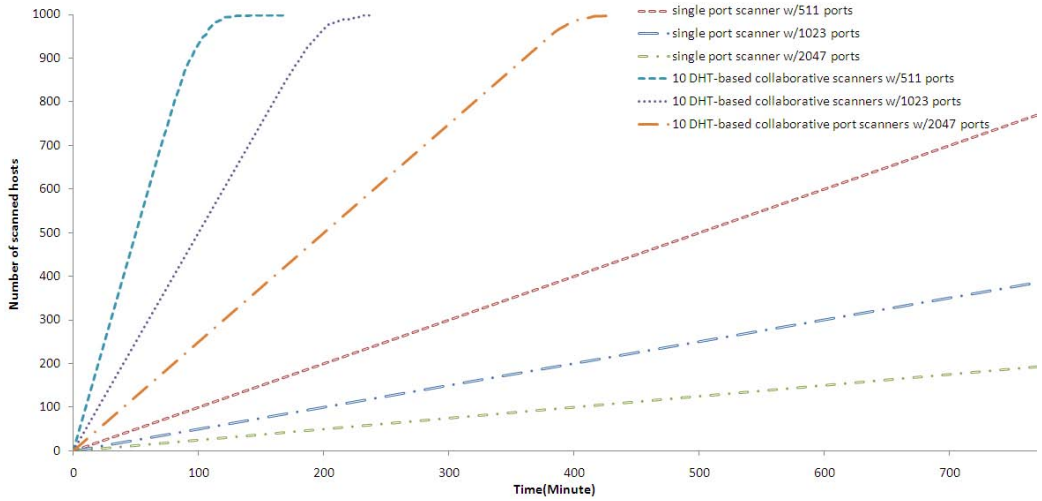

Figure 4.1.: The performance of the scanning with different scanning granularity.

4.2 Experiments on the Host Patching/Refreshing/Revisiting

We need to study how to defend the powerful collaborative attacks. One way is to refresh and patch the hosts so that the information gathered by collaborative attackers become outdated and the vulnerabilities the collaborative attackers plan to exploit no longer exist. Collaborative attackers can implement revisit policies (see Section 3.4 for detailed discussions). Questions then arise as whether the revisit policy would have a large impact on the performance of collaborative attackers and how to devise a revisit policy. Defenders to collaborative attacks could refresh themselves or intentionally disclose wrong information to foil the collaborative attack. For instance, if defenders change IP addresses every 60 minutes, the collaborative attacks could fail. In this experiment, we quantitatively measure such defense tactics. In particular, we study the impact of revisit policy on the performance of collaborative attackers. There are 20 collaborative attackers and 1,000 target hosts in this experiment. The 20 scanners are divided into 2 groups that collaborate through the DHT. Scanning of one target host takes 2 minutes to finish. For the revisit policy, making a target host revisit-able is implemented as removing its scanning status entry in the DHT. Our experimental revisit policies include:

1. Make the target hosts revisit-able after 50 minutes.
2. Make the target hosts revisit-able after 100 minutes.
3. Make the target hosts revisit-able after 150 minutes.
4. Make the target hosts revisit-able after 500 minutes.

Fig. 4.2 shows the numbers of scanned hosts with different revisit policies. The no revisit line depicts the regular collaborative scanning scheme that does not implement a revisit policy. We observe that revisit policy has a large impact on the performance of the collaborative attackers. Specifically, the revisit time significantly affects the number of scanned hosts over time. We observe that as the collaborative

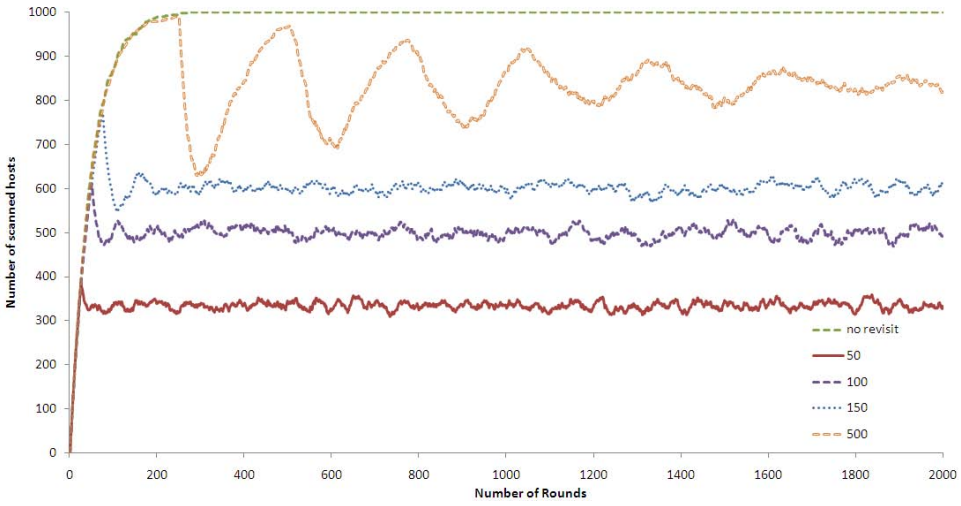

Figure 4.2.: The performance of the scanning with different revisit policies.

scan progresses, the revisit policy kicks in at a certain point, depending on the pre-set revisit time. Since a lot of hosts scanned in the beginning can "expire" according to the revisit time, the revisit policy causes a reduction in the number of hosts that the system considers as "scanned". The number of scanned hosts then fluctuates as the collaborative attackers scan the expired hosts and other hosts become expired.

Although the number of scanned hosts is not constant as the number of rounds increases, we observe that the system reaches an equilibrium around a certain number of hosts. As the revisit time increases, the equilibrium number increases as well. However, there is a tradeoff. The revisit time cannot be arbitrarily increased because longer revisit time means that the scanning results for the target hosts are less up-to-date. The revisit time cannot be arbitrarily decreased either. If the revisit time is set as the time required to perform scanning on one host, as soon as the scanning of one target host completes, the scanning results for another target host could expire and arbitrarily delay the whole port scanning. Note if the revisit time is approximately twice as much as the time required to scan all victim hosts, the system will scan all target hosts again, which renders low efficiency.

We infer some guidelines for setting up the revisit policy. If the Age Policy discussed in Section 3.4.6 is employed, the scanning latency for one target host takes time t, the size of target host space is h, the number of scanners is s, it is recommended that the system makes the target hosts revisit-able after at least $\frac{ht}{s}$ time. In our experiment, the recommended revisit time is $\frac{1000*2}{20} = 100$ (minutes), and the system is able to maintain scanning results for approximately half of all hosts. Note the Age Policy might not be the best policy. As discussed in section 3.4.6, more complex revisit policies can be employed to improve the efficiency of the system.

4.3 Experiments on Defense of Collaborative Attacks : Detection

In this experiment, our objective is to detect if the incoming attack is launched by collaborative attacks and detect collaborative attackers among all attacks.

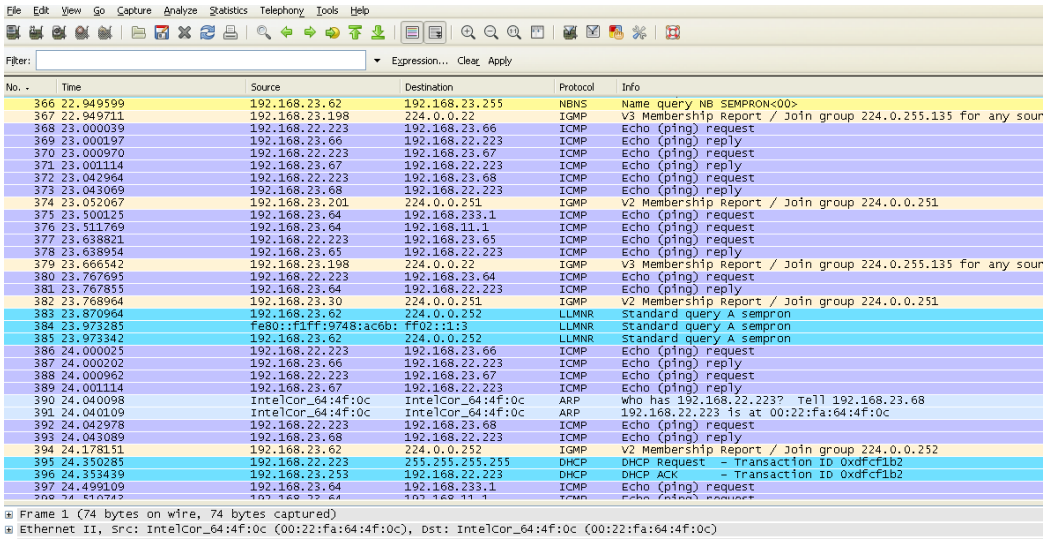

Figure 4.3.: Part of the data set used in this experiment.

Our experiment infrastructure including network of collaborative attackers. Collaborative attackers communicate (e.g., they send and receive packets from each other) and leave traces on the network. We collect data on the interaction of collaborative attackers, and combine the logs and communication data with malicious data and "peaceful" normal data to generate a huge data set. We then try to determine if the given data set contains collaborative attacks. We analyze the data to determine the collaborating groups and whether they are performing malicious activities. Fig. 4.3 produced by [10] shows an sample of the data set.

We could monitor the network gateways, routers, and switches to log essential data.

Here are the statistics for the attack data we used.

1. Average packet (/second): 16.007
2. Average packet size : 90.227 bytes
3. Average bytes/second 1444.228
4. Average MBit/sec 0.012

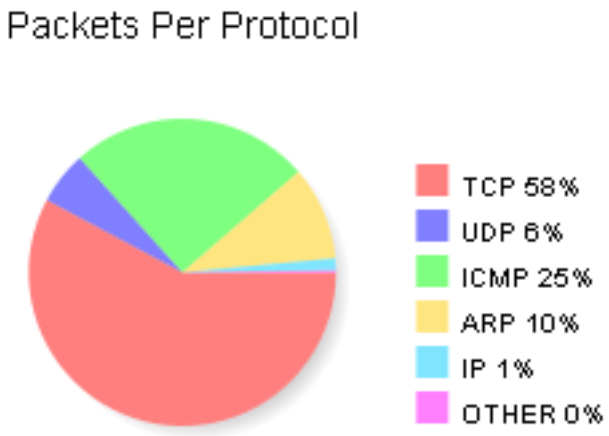

Figure 4.4.: Packet distribution of the malicious data set.

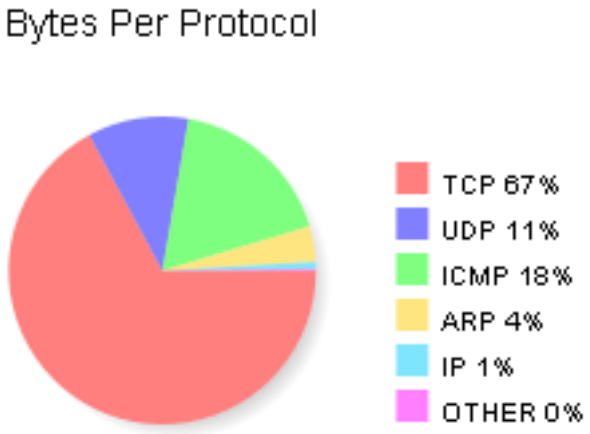

Figure 4.5.: Space distribution of the malicious data set.

5. Format: libpcap

6. Packet size limit: 65535

7. Encapsulation: Ethernet

Fig. 4.4, Fig. 4.5, and Fig. 4.6 show the packet and space distributions of the malicious data set used in our experiment. We observe that majority of the data are TCP traffic.

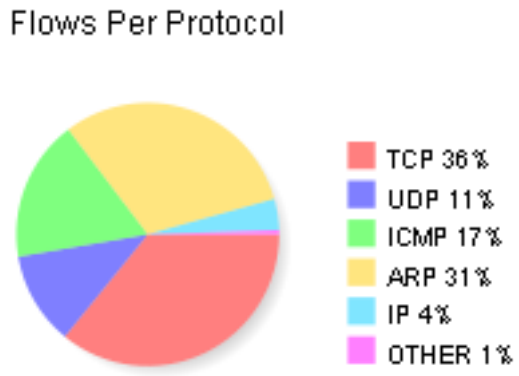

Figure 4.6.: Flow distribution of the malicious data set.

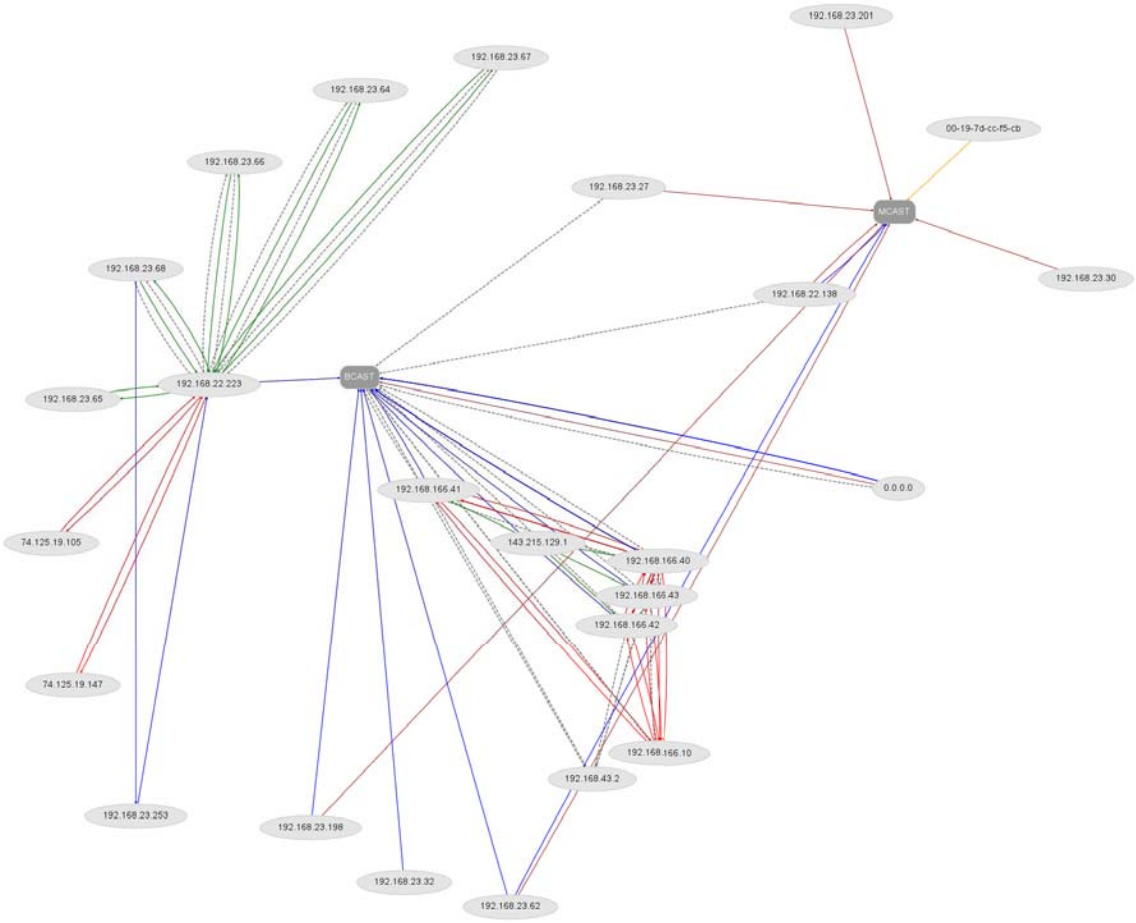

Figure 4.7.: The output of the defense analysis (grouped).

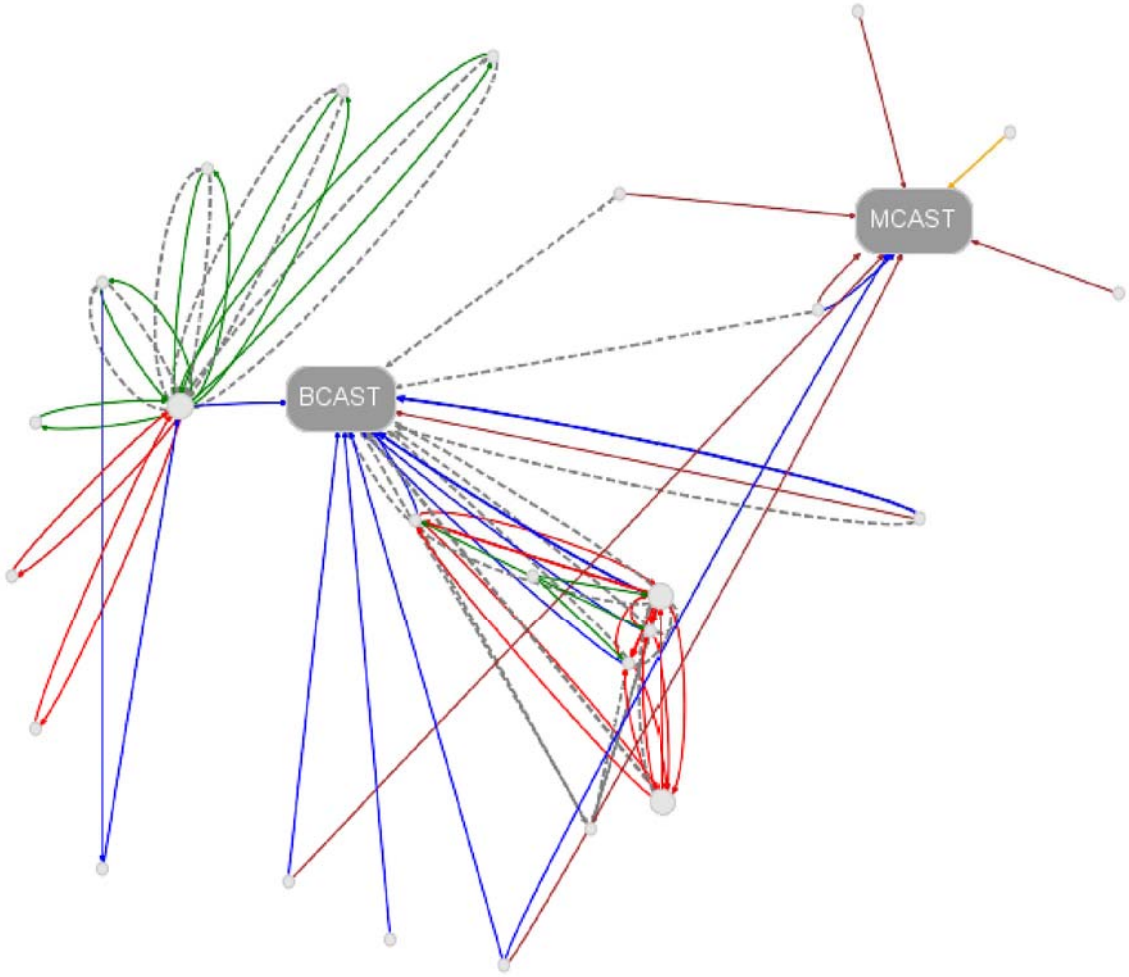

Figure 4.8.: The output of the defense analysis (Radial Model).

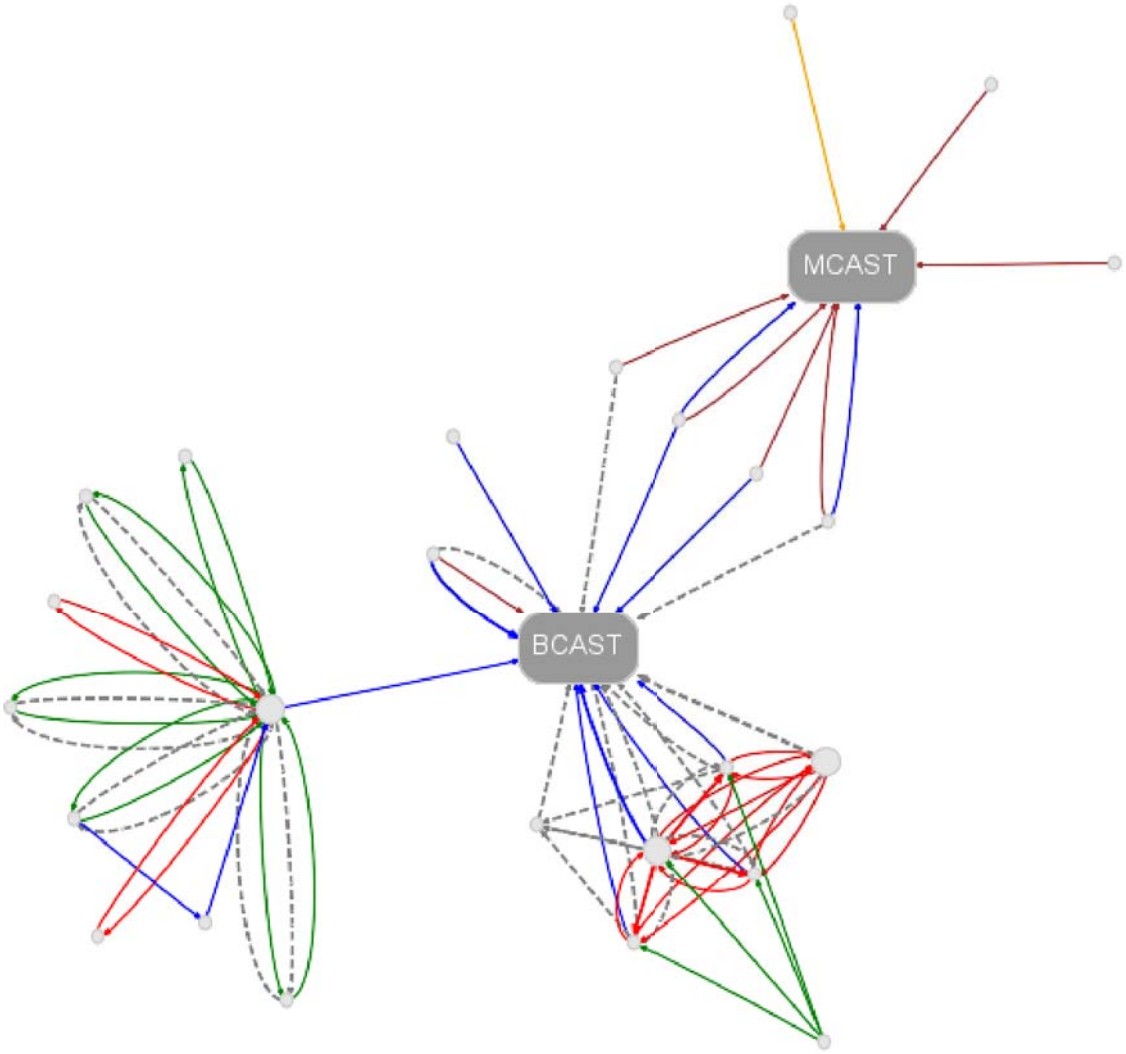

Figure 4.9.: The output of the defense analysis (Spring Model).

We try to find out collaborative groups and whether they are malicious attackers.

We use the communication graph, which represents the communication relationships among computers, and the content database, which stores the communication details among computers together to determine whether there exists collaborating attackers. More specifically, we group computers which exhibit similar communication behaviors together, and use thresholds to decide whether they are performing malicious behavior (such as excessive number of IP address lookups for collaborative port scanners). In our experiment, we set the threshold to 3, which means that the sum of the in and out degrees should exceed 3 for the collaborative attacker clique to accept a node as a member. The collaborative attacker clique must identify a core member whose sum of in and out degrees greatly exceed the normal threshold. For instance, if a collaborative clique has 4 members, the sum of in and out degrees for its core member must be no smaller than 12, and should be larger than 12 in most cases. Note that the in and out degree are relative degrees within specific LANs, hence in degree of 1 does not suggest that the node received only one message.

Models for collaboration could be flooding architecture, collaboration-server based architecture, distributed architecture, and the hybrid architecture, etc. Details regarding some of the architectures are discussed in Section 3.4.2.

Fig. 4.7 shows our experimental results. Our results show that analyzing the frequent interaction of collaborative attackers can be analyzed in communication graph and the collaboration clique can be highlighted out.

Fig. 4.8 and Fig. 4.9 show the generated communication graph by the spring and radial models used by the experiment tool SMART [84]. We clearly identify the collaborative attackers in the group. The experimental results prove the validity of the communication analysis.

We observe that we have successfully identified 80 percent of all the collaborative attackers. The threshold we set for collaborative attackers (3-edge rule) eliminated one of the collaborative attackers due to its infrequency usage of ARP messages.

5 ADDITIONAL EXPERIMENTS ON ATTACK

In this chapter, we present results on experiments on collaborative attacks and possible defense strategies. Our experiments provide insights into collaborative attack strategies, collaborative attack granularity, and defense systems such as the revisit policy. The experiment setup follows the description in previous chapters.

5.1 Experiments on the Witty Worm

We conduct experiments and compare our experimental results to the real-world propagation of the Witty Worm.

The Witty Worm [33] is suspected to employ a hitlist scanning or was released on previously hacked hosts.

Figure 1 in [33] on Witty Worm Global View (from www.caida.org), by C. Shannon and D. Moore, shows the initial spread (the first half minute) of the Witty Worm. The initial spread is unusual and could not be explained by the existing models including the AAWP model, because the worm propagated to 110 hosts in the first 10 seconds.

However, our experimental results correctly show that the effect of hitlist scanning can be approximated by the linear combination of two random scannings. When the linear Fibonacci Coefficient (h, the size of hitlist) is large, the unusual growth of the malware propagation can be explained by the linear combination of regular propagations.

Fig. 5.1 shows our experimental results with hitlist size 1 (enlarged 100 times, and right-shifted by 10 time ticks). We observe that the graph approximately matches the propagation of the Witty Worm. Subtle differences exist (e.g., the concavity), and possible causes are different operating environment and different settings of parameters (e.g., k and b).

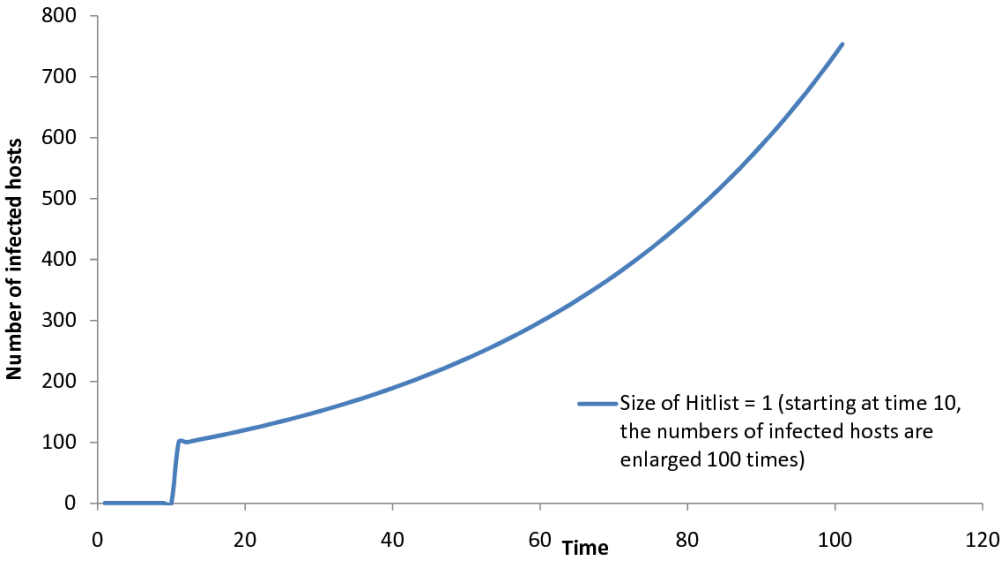

Figure 5.1.: The propagation generated by our experiments

5.2 Experiments on the Collaborative Routing and DDoS Attack

We would like to verify the existence of collaborative attacks and that they can cause more damages or gain more control of the target system. We have conducted experiments to verify the power of collaborative attacks, analyzed the collaborative attack, and generated the intrusion graph of the attack.

In the example collaborative attack, the goal is to launch a DDoS attack against a target node T, as shown in Fig. 5.2. Attackers 1, 2, .., n are directly associated with router R1 with the firewall and target node T is associated with switch S1 without a firewall. To launch DDoS attacks, attackers need to send out a large number of abnormal packets, and those packets arrive at the first router, R1, before going to the Internet.

Since R1 is a sophisticated router with a firewall, it employs a packet filtering mechanism, and can automatically filter out the incoming packets from IP addresses that are sending out large amount of abnormal traffic. Hence, regular DDoS attack packets will be filtered out and the attack will fail. However, certain vulnerabilities in router R1 can be exploited to disable its firewall and packet filtering. In a collaborative attack, one attacker can attack router R1, while other attackers launch the DDoS attack after the first one successfully disables the firewall of router R1.

Input variable parameters include:

1. N: Number of normal TCP connections;
2. M: the speed of link from each host to router, 10Mb/s;
3. B: buffer space at each router, 4K $*$ N bytes; Size_packet: packet_size, 1K bytes; and
4. MR: speed of the link between R1 and R2, 1.5Mb/s.

For the regular DDoS attack, we modify the router information controller such that router will impose a limit on the number of SYN packets per second permitted to pass. After the limit is passed router will send SYN/ACK packets for the hosts.

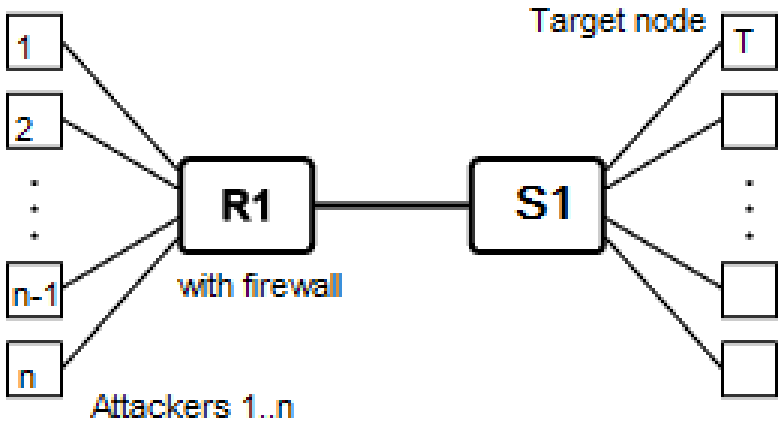

Figure 5.2.: The network topology of a hypothetical collaborative attack.

Output Performance Metrics include:

1. Round-Trip time: The time for sending a echo request and getting a reply between two nodes in the system; and

2. Bandwidth: The bandwidth of the network connection.

We used SSFNet [23], and conducted the experiments in Linux 2.6.13 with Java runtime Environment. The topology of the network is Dumbbell (Fig. 5.2).

Steps of the experiment include:

1. Initialize the system with various number of TCP connections, first with the regular DDoS attack scenario for various periods, such as 15 minutes.

2. Initialize the system with various number of TCP connections, with the collaborative DDoS and routing attack scenario for various periods, such as 15 minutes.

3. Start the system with two HTTP servers, one on each target node. The N(10) TCP connections will send traffic for 2 seconds and restart. We run the DDoS attack after 5 minutes of system start and measure estimated RTT time.

We utilize the SSF.App.DDoS package and run the DDoSSession() function. Selection of master and zombie nodes is done randomly among the nodes directly connected to Router 1(R1). Two target nodes are selected among the nodes directly connected to Router 2(R2). For the regular DDoS attack, we modify the router information controller such that router will impose a limit on the number of SYN packets per second permitted to pass. After the limit is passed router will send SYN/ACK packets for the hosts. However, for collaborative attacks, a "trojan horse" is embedded in a router. As soon as the DDoS attack is launched, the master node will send out a secret message to the router such that a "trojan" embedded in the router will change the routing information such that the router will no longer impose such SYN packet limits.

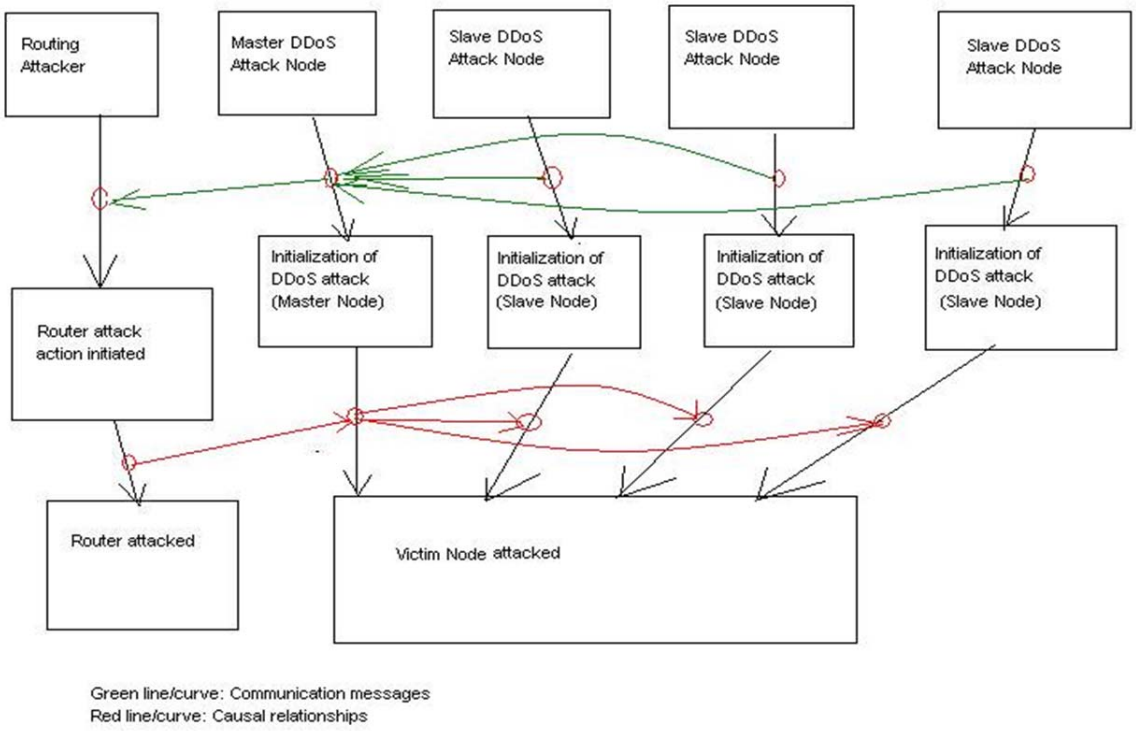

Figure 5.3.: The intrusion graph for the collaborative DDoS and routing attacks.

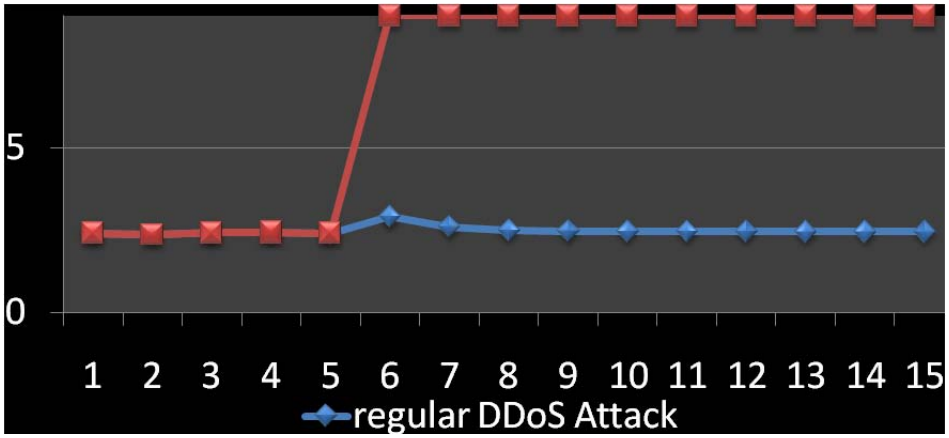

Figure 5.4.: RTT time (second) vs. time in system (min).

We analyze the collaborative DDoS and routing attacks and build the intrusion graph, shown in Fig. 5.3 (green and red arrows model the coordination between attackers).

Fig. 5.4 shows that collaborative attacks can cause much more damage than single attacks. X-axis represents time(in minutes) and Y-axis rerepresents RTT time(in seconds). In this example, the DDoS attacks were started at time t=5min. The red line shows the collaborative attacks of routing and DDoS. The Blue line shows the regular DDoS attack. Because router has the defense mechanism built-in against DDoS attack, the regular attack did not accomplish its goal. However, in the collaborative attack case, when launched together with routing attacks, DDoS attack effectively blocked the user from establishing any new TCP connection.

6 CONCLUSIONS AND FUTURE WORK

In this dissertation, we have made significant contributions to the research on collaborative attacks:

1. We study the malware propagation and present results that help understand the effects of Multi-port scanning, Multi-threading, Infection time, Multiple starting points, and Collaboration (MMIMC) on malware propagation.

2. We discuss architectures, polices, and allocations schemes for collaborative attackers. We present a fast DHT-based collaborative attack scheme that aims to eliminate duplicate attacks, minimize contention, and significantly increase the attack speed.

3. We propose different collaboration strategies and analyze their advantages and disadvantages.

4. We discuss the static, dynamic, and hybrid target selection and allocation schemes.

5. We present the algorithm details and discuss the stop and revisit policies for collaborative attackers.

6. Our experimental results provide insights into many design and implementation issues for collaborative attack and defense.

In Chapter 2, we focused on applying the Fibonacci Number Sequence (FNS) and its properties to analyze the performance of malware propagation schemes, including collaborative malware propagation. We employed the patching rate to model the

defense activities as a black box. Questions then arise as how to defend sophisticated malware and delay the collaborative propagation. Modeling and analysis of the sophisticated collaborative defense is an interesting subject for future work.

In Case 2 of Section 2.4.3, we assume that malware entities at different hosts can communicate with each other to avoid duplicate infection attempts. We note that communications may incur overhead, which could be caused by network delays, the limitations of communication protocols, the sizes of the data buffers at different hosts, etc (Currently, the overhead is not modeled). Advanced malware can also employ intelligent localized-scan algorithms. Modeling communications and processing overhead, intelligent collaboration schemes, and smart localized-scan algorithms for malware is the subject for future work. Researchers discussed how to improve the performance of scanning by sampling [17]. With the prevalence of wireless networks, there will be more dynamic hosts that may join and leave the network frequently. We plan to extend the our analysis to represent the sampling scheme and the dynamic host memberships. In Section 2.4, we assume that propagation time is the same for all infections. In the real world, propagation time for different infections may vary. If the propagation time is three time slices, we can apply the Tribonacci Number Sequence [31] to study the malware propagation. Analysis of effects of varying propagation times is the subject for future work.

There are a number of ways to enhance or defend the collaborative attack scheme discussed in Chapter 3, which are the subjects for future work.

First, the port scanners can employ the insider collaboration technique. In the insider collaboration attack, an insider gathers the knowledge about vulnerable hosts. The outsider launches port scanning with the pre-acquired knowledge from the insider. In this case, the knowledge of vulnerable hosts can be gathered offline rather than online.

Second, port scanners can also employ heuristics and more intelligent algorithms. For example, learning algorithms can be utilized. However, such port scanners may scan very slowly due to the complicity of the algorithms. Solutions include offline

training of the scanner with a lot of log data. Smarter revisit policies can be employed as well.

Third, collaborative port scanners can employ the passive and stealth technique. For example, they can passively log and analyze the network traffic. Such techniques could enhance the collaborative port scanners and challenges the defense systems.

Finally, we could fingerprint the collaboration methods employed by the collaborative attackers. Defense systems can implement algorithms that learn and classify the communication patterns like the flooding, server-based, and distributed architectures. Then, they can monitor and analyze the network traffic to detect these collaboration patterns and flag corresponding nodes as possible collaborative attackers.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Y. Zhang and B. Bhargava, The Effects of Threading, Infection Time, and Multiple-Attacker Collaboration on Malware Propagation, The 28th IEEE International Symposium on Reliable Distributed Systems (SRDS), Niagara Falls, NY, Sep. 2009

[2] B. Bhargava, Y. Zhang, N. Nidika, L. Lilien, and M. Azarmi, Collaborative attacks in WiMAX networks, WILEY International Journal of Security and Communication Networks (SCN), Special Issue, 2009.

[3] S. Friedl, Analysis of the New Code Red II Variant, `http://www.unixwiz.net/techtips/CodeRedII.html`, Last accessed Apr. 3, 2009

[4] `http://blogs.pcmag.com/securitywatch/2009/11/jailbroken_iphones_hacked_user.php`

[5] R. Vogt, J. Aycock, and M. Jacobson, Army of Botnets, Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, Feb. 2007

[6] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. Proceedings of the 7th Symposium on Operating System Design and Implementation, 2006.

[7] Burton H. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Vol. 13, 1970

[8] C. Zou, W. Gong, and D. Towsley, Code Red Worm Propagation Modeling and Analysis, Proceedings of the 9th ACM Conference on Computer and Communication Security , Washington D.C., Nov. 2002

[9] D. Moore, C. Shannon, and J. Brown, Code-Red: a Case Study on the Spread and Victims of an Internet Worm, Proceedings of ACM/USENIX Internet Measurement Workshop, France, Nov. 2002

[10] `http://www.wireshark.org/`

[11] D. Dagon, G. Gu, C. Lee, and W. Lee, A Taxonomy of Botnet Structures, Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC), Dec. 2007.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's Highly Available Key-Value Store, Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles, Oct. 2007

[13] Editorial, A Cyberblockade in Estonia, New York Times, Jun. 2, 2007

[14] T. Koshy, Fibonacci and Lucas Numbers with Applications, Wiley-Interscience, Aug. 2001

[15] A. Svensson, A Note on Generation Times in Epidemic Models, Mathematical Biosciences, Vol. 208, Iss. 1, Jul. 2007

[16] Kademlia Specification, `http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html`, last accessed Jul. 1, 2009

[17] M. Vojnovic, V. Gupta, T. Karagiannis, and C. Gkantsidis, Sampling Strategies for Epidemic-Style Information Dissemination, Proceedings of the IEEE INFOCOM, Apr. 2008

[18] Z. Chen, L. Gao, and K. Kwiat, Modeling the Spread of Active Worms, Proceedings of the IEEE INFOCOM, Apr. 2003

[19] NMAP documentation, /urlhttp://nmap.org/book/man-performance.html, last accessed Jul. 1, 2009

[20] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, DDoS Defense by Offense, ACM SIGCOMM 2006, Pisa, Italy, Sep. 2006

[21] `http://www.sans.org/reading_room/whitepapers/vpns/the_day_des_died_722`

[22] D. Moore, G. M. Voelker, and S. Savage, Inferring Internet Denial-of-Service Activity, Usenix Security Symposium, 2001.

[23] `http://www.ssfnet.org/homePage.html`

[24] S. Staniford, V. Paxson and N. Weaver, How to Own the Internet in Your Spare Time, Proceedings of the 11th USENIX Security Symposium, Aug. 2002

[25] A.G. Voyiatzis and D.N. Serpanos, Pulse: A Class of Super-Worms Against Network Infrastructure. Proceedings of ICDCS Workshops, May 2003

[26] Z. Chen and C. Ji, A Self-Learning Worm Using Importance Scanning, ACM Workshop on Rapid Malcode, Nov. 2005

[27] A. Wagner, T. Dubendorfer, B. Plattner, and R. Hiestand, Experiences with Worm Propagation Simulations, Proceedings of ACM Workshop on Rapid Malcode, Oct. 2003

[28] S. Sarat and A. Terzis, Measuring the Storm Worm Network. Technical Report 01-10-2007, `http://hinrg.cs.jhu.edu/uploads/Main/STORMTR.pdf`

[29] C. Kanich, K. Levchenko, and B. Enright, G. M.Voelker and S. Savage, The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff, Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Franciso, CA, Apr. 2008

[30] Y. Zhang and B. Bhargava, Fibonacci Modeling of Malware Propagation, Technical Report TR-08-017, Department of Computer Sciences, Purdue University, 2008

[31] I. Dumitriu, On Generalized Tribonacci Sequences and Additive Partitions, Discrete Mathematics, Vol. 219 , Iss. 1-3, 2000

[32] Z. Chen and C. Ji, Optimal Worm-Scanning Method Using Vulnerable-Host Distributions International Journal of Security and Networks, Special Issue on Computer and Network Security, Vol. 2, 2007

[33] http://www.caida.org/research/security/witty/, last accessed Jul. 1, 2009

[34] C. Zou, D. Towsley, and W. Gong, On the Performance of Internet Worm Scanning Strategies, Performance Evaluation, Jul. 2006

[35] J. Yang, Fast Worm Propagation in IPv6 Networks, http://www.cs.virginia.edu/~jy8y/publications/cs85104.pdf

[36] http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

[37] J. Ma, G. Voelker and S. Savage, Self-stopping Worms, Proceedings of the ACM Workshop on Rapid Malcode (WORM), Washington D.C., Nov. 2005.

[38] R. Vogt, J. Aycock, and M. Jacobson Jr., Quorum Sensing and Self-Stopping Worms. Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM 2007), Alexandria, VA, Nov. 2007.

[39] Detecting and Recovering from a Virus Incident http://www.sans.org/reading_room/whitepapers/malicious/903.php

[40] G. Gu, J. Zhang, and W. Lee, BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2008

[41] B. Wiley, Curious Yellow: The First Coordinated Worm Design, http://blanu.net/curious_yellow.html, Last Accessed Apr. 20, 2008

[42] C. Zou, D. Towsley, and W. Gong. On the Performance of Internet Worm Scanning Strategies, Elsevier Journal of Performance Evaluation, Jul. 2006

[43] J. Wu, S. Vangala, L. Gao, and K. Kwiat, An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques, Network and Distributed System Security Symposium (NDSS), 2004

[44] J. Twycoss, M. Williamson: Implementing and Testing a Virus Throttle. Proceedings of the 12th USENIX Security Symposium, Washington, 2003

[45] M. Vivo, E. Carrasco, G. Isern, G. Vivo, A Review of Port Scanning Techniques, ACM Computer Communications Review, Vol. 29, Apr. 1999

[46] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, Survey and Taxonomy of IP Address Lookup Algorithms, IEEE Network Magazine,vol.15, Mar.-Apr. 2001

[47] J. Jung, V. Paxson, A. Berger, and J. Balakrishnan, Fast Portscan Detection Using Sequential Hypothesis Testing, Proceedings of the IEEE Symposium on Security and Privacy, May 2004

[48] S. Staniford, J. Hoagland, J. McAlerney, Practical Automated Detection of Stealthy Portscans. Journal of Computer Security 10(1/2), 2002

[49] S. Bellovin, B. Cheswick, A. Keromytis, Worm Propagation Strategies in an IPv6 Internet. http://www.cs.columbia.edu/~smb/papers/v6worms.pdf, LOGIN, Vol 31. No.1.

[50] P. Wang, S. Sparks, and C. Zou, An Advanced Hybrid Peer-to-Peer Botnet, preprint, IEEE Transactions on Dependable and Secure Computing, 2009

[51] A. Kamra, H. Feng, V. Misra and A. Keromytis, The Effect of DNS Delays on Worm Propagation in an IPv6 Internet, Proceedings of IEEE Infocom, Miami, FL, 2005.

[52] A. Kumar, V. Paxson, N. Weaver, Exploiting Underlying Structure for Detailed Reconstruction of an Internet-scale Event, Proceedings of ACM IMC, New Orleans, LA, Oct. 2005.

[53] C. Gates, Coordinated Port Scans: A Model, A Detector and An Evaluation Methodology. Ph.D. Thesis. Dalhousie University. Feb., 2006

[54] http://www.bittorrent.com/

[55] http://www.emule-project.net/

[56] http://www.distributed.net/

[57] Strange Attractors and TCP/IP Sequence Number Analysis – One Year Later, http://www.iu.hio.no/~haugerud/ids/SAATSNA_OYL.pdf

[58] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, Looking Up Data in P2P Systems, Communications of the ACM, Feb. 2003.

[59] J. Cho and H. Garcia-Molina, Effective Page Refresh Policies for Web Crawlers, ACM Transactions on Database Systems, 28(4): Dec. 2003.

[60] B. Bhargava and C. Hua, A Causal Model for Analyzing Distributed Concurrency Control Algorithms, IEEE Transactions on Software Engineering, 1983

[61] L. Lamport, Time, Clocks and the Ordering of Events in a Distributed System, Communications of the ACM 21.7, 558-565, Jul. 1978.

[62] R. Lippmann and K. Ingols, An Annotated Review of Past Papers on Attack Graphs. Technical report, MIT Lincoln Laboratory, Mar. 2005.

[63] X. Li and S. Xu, A Stochastic Modeling of Coordinated Internal and External Attacks. Technical Report available at http://www.cs.utsa.edu/~shxu/collaborative-attack-model.pdf

[64] S. Ramaswamy, H. Fu, and K. E. Nygard, Effect of Cooperative Black Hole Attack on Mobile Ad Hoc Networks, International Conference on Wireless Networks, 2005

[65] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, An On-Demand Secure Routing Protocol Resilient to Byzantine Failures. In ACM Workshop on Wireless Security (WiSe) in conjunction with MobiCom, 2002

[66] A. Hussain, J. Heidemann, and C. Papadopoulos, COSSACK: Coordinated Suppression of Simultaneous Attacks, In DISCEX, 2003

[67] D. Ourston, S. Matzner, W. Stump, and B. Hopkins, Coordinated Internet Attacks: Responding to Attack Complexity, Journal of Computer Security. Vol. 12(2), 2004

[68] F. Cuppens and A. Miege, Alert Correlation in a Cooperative Intrusion Detection Framework, IEEE Symposium on Security and Privacy, 2002

[69] Jiahai Yang, Peng Ning, X. Sean Wang, and Sushil Jajodia, CARDS: A Distributed System for Detecting Collaborative Attacks. Proceedings of IFIP TC11 Sixteenth Annual Working Conference on Information Security, Aug. 2000.

[70] J. Garca, F. Autrel, J. Borrell, Y. Bouzida, S. Castillo, F. Cup-pens et G. Navarro, Preventing Collaborative Attacks Via Alert Correlation, 9th Nordic Workshop on Secure IT Systems (NORDSEC 2004). Finland, Nov. 2004.

[71] Sanjay Ramaswamy, Huirong Fu, Manohar Sreekantaradhya, John Dixon, and Kendall Nygard, Prevention of Cooperative Black Hole Attack in Wireless Ad Hoc Networks, ICWN'03, Las Vegas, NV, USA, Jun. 2003.

[72] J. Douceur, The Sybil Attack, 1st International Workshop on Peer-to-Peer Systems, 2002.

[73] A.Ramachandran and N.Feamster, Understanding the Network-Level Behavior of Spammers, In ACM SIGCOMM, Vol. 36, 2006.

[74] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, Inside the Slammer Worm. IEEE Security and Privacy journal, Vol. 1, 2003

[75] D. Moore, C. Shannon, J. Brown, Code-Red: a Case Study on the Spread and Victims of an Internet Worm, ACM/USENIX IMW, 2002.

[76] S. Katti, B. Krishnamurthy and D. Katabi, Collaborating Against Common Enemies. ACM Internet Measure Conference (IMC), 2005.

[77] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, D. Zamboni. Analysis of a Denial of Service Attack on TCP. IEEE Symposium on Security and Privacy, 1997

[78] D. Pei, D. Massey, and L. Zhang, Detection of Invalid Routing Announcements in the RIP Protocol, GLOBECOM, 2003

[79] S. Cheung, U. Lindqvist, and M. Fong, Modeling Multistep Cyber Attacks for Scenario Recognition, DARPA Information Survivability Conference and Exposition, 2003.

[80] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, SybilGuard: Defending Against Sybil Attacks via Social Networks, Proceedings of ACM SIGCOMM Conference, Sep. 2006

[81] H. Yu, P. B. Gibbons, and M. Kaminsky, Toward an Optimal Social Network Defense Against Sybil Attacks, Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, 2007

[82] http://www.dshield.org/

[83] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, OpenDHT: A Public DHT Service and Its Uses, Proceedings of ACM SIGCOMM 2005, Aug. 2005.

[84] Safe Mapping and Reporting Tool (SMART), `http://safemap.sourceforge.net/`

VITA

VITA

Yu Zhang received the B.E. degree in computer science, at the age of 19, from Special Class for Gifted Young, University of Science and Technology of China, the M.S. degree in computer science, at the age of 21, from Purdue University, and the Ph.D. degree in computer science, at the age of 24, from Purdue University. He has worked at Cisco Systems, VMware, and Google. He has published in various journals and conferences. He is a recipient of a Computer Science Graduate Student Board travel grant for the IEEE Symposium on Reliable Distributed Systems. Part of this dissertation has appeared in research proposals and journal papers.