Topology-Aware Vulnerability Mitigation Worms

Defensive Worms

Ziyad S. AL-Salloum

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Royal Holloway
University of London

2011

Topology-Aware Vulnerability Mitigation Worms

Information Security Group
Royal Holloway, University of London

Dedication

قُلۡ إِنَّ صَلَاتِي وَنُسُكِي وَمَحۡيَايَ وَمَمَاتِي لِلَّهِ رَبِّ ٱلۡعَٰلَمِينَ ١٦٢

. . . Indeed, my prayer, my rites of sacrifice, my living and my dying are for Allah , Lord of the worlds.

–The Holy Quran [Al-Anam, Verse 162]

Declaration of Authorship

I, Ziyad S. AL-Salloum, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated. This work has not been submitted for any other degree or award in any other university or educational establishment.

Ziyad S. AL-Salloum

December, 2011

Do not go where the path may lead;
go instead where there is no path and
leave a trail.

Ralph Waldo Emerson [1803 – 1882]

Preface

Passion to research is like soul to body, it brings it to life and drives it. But it is hard to be passionate about something that can not go beyond the paper it has been written on. Researchers – I believe – have the noble duty of raising the people's quality of life by proposing solutions to their problems; dont waste your time on something else.

Yet, some problems are hard to solve and the root cause, might be, that we think within the box we live in. Breaking that box, might be necessary, to bring new research paths up to the surface and help push the research community a bit forward; indeed if no one thought differently in the past, we would not enjoy what we have in the present.

Free yourself from anything you take for granted, release your imagination, and start changing. They will ignore you, laugh at you, then fight you, but if you are patient enough you will get through.

Acknowledgments

I would like to thank God, the Most Gracious, the Most Merciful for accomplishing my doctoral studies. I would like to thank my family and my government for sponsoring and supporting me.

I am also grateful to my supervisor, Dr. Stephen D. Wolthusen, for maintaining high standards in supervision, academic knowledge, and attitude. I have to admit that my attempts to find something he is not aware of in the broad field of information security have failed. In addition, I would like to thank my advisor Dr. Chez Ciechanowicz for his support, Dr. Michael Meier for his valuable comments, and Dr. Allan Tomlinson for taking the extra step in providing insightful views and comments on different parts of my work.

I am also grateful to Viet Pham, Penying Rochanakul, and Saif Al-Kuwari, for the nice comments they provided to enhance the readability of some individual chapters. I am also deeply grateful to all the lecturers, secretaries and students in the math department for providing me with a healthy and productive research environment during my Ph.D. studies.

I also like to acknowledge the campus security guards (especially night shifts) for making researching after midnight peaceful and quiet. And I will not forget to express my gratitude to all those I have met at RHUL prayer room, indeed that humble room became a busy hub to meet different students on campus year after year. Lastly, I would like to thank Eng. Sami El-Taji for remembering me in his prayers and – the extraordinary kind – Bitty (my neighbor) whom always cared.

Abstract

In very dynamic Information and Communication Technology (ICT) infrastructures, with rapidly growing applications, malicious intrusions have become very sophisticated, effective, and fast. Industries have suffered billions of US dollars losses due only to malicious worm outbreaks. Several calls have been issued by governments and industries to the research community to propose innovative solutions that would help prevent malicious breaches, especially with enterprise networks becoming more complex, large, and volatile.

In this thesis we approach self-replicating, self-propagating, and self-contained network programs (i.e. worms) as vulnerability mitigation mechanisms to eliminate threats to networks. These programs provide distinctive features, including: Short distance communication with network nodes, intermittent network node vulnerability probing, and network topology discovery. Such features become necessary, especially for networks with frequent node association and disassociation, dynamically connected links, and where hosts concurrently run multiple operating systems.

We propose – to the best of our knowledge – the first computer worm that utilize the second layer of the OSI model (Data Link Layer) as its main propagation medium. We name our defensive worm *Seawave*, a controlled interactive, self-replicating, self-propagating, and self-contained vulnerability mitigation mechanism. We develop, experiment, and evaluate Seawave under different simulation environments that mimic to a large extent enterprise networks. We also propose a threat analysis model to help identify weaknesses, strengths, and threats within and towards our vulnerability mitigation mechanism, followed by a mathematical propagation model to observe Seawave's performance under large scale enterprise networks. We also preliminary propose another vulnerability mitigation worm that utilizes the Link Layer Discovery Protocol (LLDP) for its propagation, along with an evaluation of its performance.

In addition, we describe a preliminary taxonomy that rediscovers the relationship between different types of self-replicating programs (i.e. viruses, worms, and botnets) and redefines these programs based on their properties. The taxonomy provides a classification that can be easily applied within the industry and the research community and paves the way for a promising research direction that would consider the defensive side of self-replicating programs.

Contents

Abbreviations

ARP	Address Resolution Protocol
BBN	Bayesian Belief Network
BPDU	Bridge Protocol Data Unit
CAM	Content Addressable Memory
DoS	Denial of Service
IP	Internet Protocol
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
LSD	Link State Database
MAC	Message Authentication Code
Mbps	Mega bits per second
MIB	Management Information Base
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
RARP	Reverse Address Resolution Protocol
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol

Chapter 1

Introduction

1.1 Motivation

In the world of Information and Communication Technologies (ICTs), network applications are evolving rapidly and so are their threats. Malicious worms and botnets have been identified as one of the most significant threats facing the industry; indeed their outbreaks have cost the industry billions of US dollars. Furthermore, it has been observed that the majority of malicious worm outbreaks have utilized a publicly known vulnerability with a vendor level patch already available. If security administrators were able to identify and reach each vulnerable node in a rapid manner and mitigate the vulnerability, then many malicious attacks would have been prevented. Therefore, calls have been raised by industries and governments to search for innovative solutions to respond to malicious worms attacks. For example the chief of U.S. Department of Homeland Security have indicated that ″The key thing we learnt from Stuxnet was the need for rapid response across the private sector.″ She also noted that recent malicious attacks have reached a high level of sophistication and novelty [61].

Different difficulties prevent security administrators from providing immediate treatment to each single vulnerable node in the enterprise network. However – in general – these obstacles raise from the lack of ability to instantly interact with each network device, due to some reasons, including:

- **Dynamic nature of Internetworking.** Enterprise networks are becoming larger, complex, and volatile by nature. Where it is common for subnets to emerge and disappear bound to business, technical, or administrative demands, in an environment where network nodes associate and disassociate dynamically. This makes it difficult for network administrators to draw a detailed picture of their network to better manage its risks and vulnerabilities.

- **Transient Connectivity.** In real world networks, links are not always active. Connectivity is transient, based on how network links are implemented or due to unforeseen failures making it difficult to determine current active network paths.
- **Use of Virtual Machines that are Connected Intermittently.** Many nodes host multiple operating systems at different times (or at the same time); which makes it difficult to determine which OS is currently running for effective assessment.
- **Lack of Resources.** Traditional vulnerability assessment techniques for large-scale networks require more resources and are not cost effective, which leaves many organizations (especially non-profit ones) fall short in obtaining the required resources to provide efficient protective measures to their networks.

The industry – up until now – has failed to provide efficient, reliable, and rapid vulnerability mitigation mechanisms that give an instant, accurate, and detailed vulnerability map of the enterprise network. Such a map would provide the means for the security team to take action and interact with target nodes to overcome potential threats promptly. We believe that controlled, self-replicating, self-propagating, and self-contained network programs (or defensive worms) can provide such functionality; since – in some of its applications – defensive worms are not meant to be an external short term solution installed in the network to overcome an uncommon problem, but rather an immune system, running within the network providing constant monitoring and continues protection. However, many challenges face the consideration of worms as commercial solutions in the ICT industry. To the best of our knowledge there is no solution in the industry that adopts this technology, and in this thesis we try to overcome these obstacles, by trying to answer the following research questions:

- Does utilizing the features of self-replicating, self-propagating, and self-contained network programs (worms) provide rapid and effective vulnerability mitigation coverage?

- Does utilizing topology information and communication between agents aid in controlling the propagation of self-replicating, self-propagating, and self-contained vulnerability mitigation network programs?

1.2 Summary of Contributions and Organization of the Thesis

Our main contributions to the body of knowledge in the field of self-replicating, self-propagating, and self-contained network programs (worms), include:

- We revisit self-replicating programs (viruses, worms, and botnets) definitions and propose new ones to include the defensive prospective. We also propose design guidelines for defensive worms.

- We propose a novel controlled, topology-aware, interactive, self-replicating, self-propagating, and self-contained network vulnerability mitigation system (or vulnerability mitigation worm), that utilizes CAM and STP information to propagate. To the best of our knowledge the system is the first computer worm that uses layer two of the OSI model as its main propagation medium.

- Based on STP, CAM, ARP, and OSPF, we further enhance and improve our defensive worm by adding edge node failure recovery, network backbone traversal, and intermittent node detection and recovery.
- We propose another simple but novel vulnerability mitigation worm that propagates with only knowledge of immediate network neighborhood as can be obtained from passive observations of the LLDP protocol.
- We observe and evaluate Seawave's performance in response to a malicious random scanning worm outbreak. We also discuss mechanisms to protect Seawave against subversion and ensure the confidentiality and integrity of its communications.
- We propose a threat analysis model based on Bayesian Belief Networks to analyze and quantify threats towards our vulnerability mitigation mechanism.
- We propose and analyze an analytical propagation model of our defensive worm, to observe its performance in large-scale enterprise networks.

We hope our contributions joined by other's work in that not yet well-founded field of study, would help in building a foundation that paves the way for this topic to become well researched in academia.

In the remaining of the thesis, we give in Chapter 2 an overview of defensive worms where we redefine viruses, worms, and botnets, based on their attributes and free from any prejudgments. Then, we further discuss defensive worms by highlighting related work that covered the use of worms for beneficial purposes in Chapter 3. We then introduce our vulnerability mitigation worm (Seawave) in Chapter 4 and further improve it in Chapter 5, followed by another vulnerability mitigation worm based on LLDP in Chapter 6; before releasing Seawave to overcome a malicious random scanning worm in Chapter 7. We then propose a threat analysis model to assess the risks towards Seawave in Chapter 8, followed by a mathematical propagation model to evaluate Seawave in large-scale enterprise networks in Chapter 9. Our summary, conclusions, and future work then follows in Chapter 10.

1.3 Publications

The material of this thesis contains previously published papers, all with my academic supervisor Dr. Stephen D. Wolthusen, as follows:

- Chapter 4 [4]
- Chapter 5 [5]
- Chapter 6 [3]
- Chapter 7 [6]
- Chapter 8 [8]
- Chapter 9 [7]

Chapter 2

Defensive Worms – An Overview

2.1 Introduction

Several network interruptions, over the past few years, have been observed as the result of malicious worms. Businesses that rely heavily on the Internet have suffered serious financial losses, due to continuing Internet attacks [22]. For example, malicious worms such as SQL Slammer, Code-Red, and Conficker have cost the industry 1.2, 2.6, and 9.1 Billion of US dollars, respectively [54, 66, 84]. The evolvement of botnets as a major source of cybercrime (an industry worth more than 10 Billion of US dollars [34]) has also added to the challenges; owing to the difficulty of allocating and disinfecting malicious bots [9, 62]. These have led the ICT industry to take information security more seriously and start addressing innovative solutions to help evade current network threats, especially with attacks becoming more sophisticated, well targeted, and increasing in volume [60, 34] – about one third of European Internet users reported a security incident in 2010 [36]. However, this is not a straightforward task, as in today's networks the task of monitoring and managing assets has become more challenging; especially within more complex, large, and volatile enterprise networks. The challenges further increase when devices associate and disassociate frequently, along with links connecting dynamically in a network where hosts occasionally run multiple operating systems, leaving security administrators in an often unpredictable environment.

It is worth noticing that most malicious worm hits have utilized vulnerabilities that are publicly known, which indicates that the time-window between the announcement of a vulnerability and its exploitation is too short for effective patch deployment. Table 2.1 shows the interval between vulnerability announcement and worm appearance.

Name	Vulnerability Announcement	Worm Observed	Interval– Days
Code Red	26 June, 2001	12 July, 2001	16
Slapper	30 July, 2002	14 Sept. 2002	45
SQL Slammer	24 July, 2002	25 January, 2003	185
Blaster	16 July, 2003	11 August, 2003	26
Zotob	9 August, 2005	16 August, 2005	7
Conficker.A	23 October, 2008	21 Nov. 2008	29
Conficker.B	//	29 Dec. 2008	67
Conficker.B++	//	20 Feb 2009	120
Conficker.E	//	7 April 2009	168
Stuxnet	// (MS08-067)	17 June 2010	602

Table 2.1: The number of days between the announcement of a *wormable* vulnerability (with a patch) and the worm appearance [11, 66, 63, 23, 71, 70] – Stuxnet has used different vulnerabilities to propagate.

As it appears from table 2.1 worms such as Code Red, Blaster, and Conficker.A took 16, 26, and 29 days respectively after vulnerability announcement to breakout – not giving enough time for system administrators to deploy patches. Perhaps, the delay in distributing updates might be explained by the quality of security-related configurations, but the question of how efficient current mitigation mechanisms are in providing sound protection to enterprise networks remains. The ability to mitigate these vulnerabilities prior to worm outbreaks would save the industry billions of US dollars. Yet – in the real world – effective, quick, and orderly patching that ensures network systems are always up-to-date with the latest patches, is hard to obtain. As in addition to the challenges addressed perviously, system administrators need to test and examine patches before installing them – to avoid inconsistency and service disruption – which delays patch deployment. The time it takes for patch examination depends mostly on the magnitude of the patches to be installed – sometimes organizations become overwhelmed with the amount of patches they test [18]. Furthermore, usually installing patches involves rebooting the system, leaving servers where the service uptime is crucial vulnerable, until system administrators find a way to tolerate service disruption [106]. Therefore, there exists a need for vulnerability mitigation mechanisms that can tackle these problems and can at least fill the gap between wormable vulnerability discovery and malicious worm outbreaks.

HP Labs borrowed some worm techniques to distribute fixes within their network; it helped them avoid the hit of the malicious worm Blaster, saving them a large financial loss. "Our countermeasure code took a very similar approach to the actual Blaster worm," said HP; however the payload transferred via TFTP to the infected machine was a "remediation code." Therefore, before the outbreak of Blaster, "HP had patched or disabled a huge number of their [vulnerable] machines" [16]. However, HP chose not to use worm propagation but rather distributed scanners or exploiters in a way that mostly mimics worm's behaviors. The company avoided using an actual worm as a precaution due to managerial fears that the worm might get out of control, thus adding significant overhead and complexity to the implementation of such an approach, as well as exposing the network to the limitations of scanners[1]. Meanwhile, HP managed to circumvent the delay usually associated with patch deployment by distributing temporary remediations, where a formal patch is yet to be installed – when time permits. They have also avoided violating the privacy law by exploiting their own systems.

Defensive worms can be used to fill the gap between vulnerability exposure and patching, in the same way as HP did. However, worms are hard to monitor and control, and usually generate high amount of traffic that might cause denial of service attacks and lead to network congestion – SQL Slammer, for example, has consumed very high bandwidth that blocked Internet access in South Korea [89]. But these problems are not insoluble; research in

[1]Scanners limitations are covered later in this chapter.

this field can always improve our understanding of worms and give us the ability to utilize them to further protect enterprise networks.

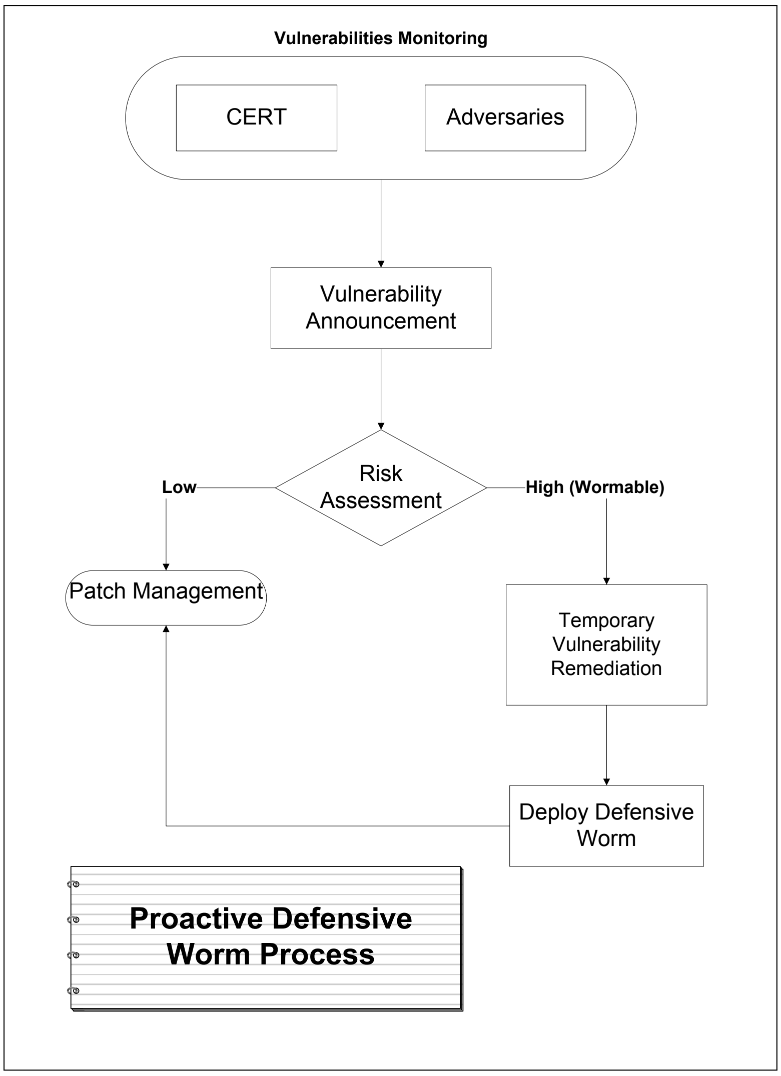

Vulnerabilities Monitoring
CERT
Adversaries
Vulnerability Announcement
Risk Assessment
Low
High (Wormable)
Patch Management
Temporary Vulnerability Remediation
Deploy Defensive Worm
Proactive Defensive Worm Process

Figure 2.1: Proactive defensive worm incident response procedure.

For more effective vulnerability mitigation procedures, the ability to know which vulnerability is wormable (i.e. has the potential to be utilized by a worm) and how to temporarily remediate it becomes necessary – models to evaluate how wormable is a vulnerability exist [73] as well as short-time remediation techniques [106]. For example, a nonprofit international organization for vulnerability assessment can be formed to address these issues and alert security administrators to take precautionary measures; and upon alert a defensive worm can be released to mitigate that vulnerability, before a malicious worm or an intruder exploit it. Based on HP's threat management procedure [16], Fig. 2.1 shows a possible process of a wormable vulnerability incident response.

Unfortunately, it can be noticed that the dominating opinion and the initial impression usually associated with worms have always been linked to malicious intent [12]. Indeed, the observed malicious worm attacks in recent years and the lack of comprehensive research on the beneficial side of these network programs have contributed to that negative view. However, if we released ourselves from any prejudgments regarding worms and tried to view them as an information distribution technique that may provide some distinctive features, that include [2]:

- **Short distance communication with target nodes.** Which speeds up interactions, decreases the probability of link and host-to-host communication failures, and keeps the bandwidth generated between the worm and its target away from main network links.

- **Intermittent node vulnerability detection.** Enables the discovery of offline hosts and portable devices (i.e. laptops, smart phones) once they have been associated with the network.

- **Network topology discovery[2].** Assesses in the discovery of undocumented nodes, subnets, and other network devices that network administrators might not be aware off.

- **Intelligent Network Propagation.** Enables the traversal of the network according to security administrators preferences or task requirements.

[2]This feature becomes useful in detecting malicious botnets.

- **No single point of failure.** The collection of agents form the scope of the worm and the failure of an agent has minimum impact on the total scope of work.
- **Workload distribution.** The impact of the workload is split between the nodes within the scope of work.

Perhaps, then, we may be able to form a more comprehensive opinion on this controversial topic of research.

2.2 A Taxonomy of Viruses, Worms, and Botnets

In general, to be able to consider the beneficial side of self-replicating programs and to easily classify them, it is necessary to observe this family of programs from a different perspective and revisit their definitions to pave the way for a research direction that would consider the defensive side of self-replicating programs. When we observe self-replicating code based on its properties, we might be able to obtain a less biased view of its nature. In this section we revisit the common negative conception usually associated with viruses, worms and botnets[3] and observe how these programs relate to each other providing a classification that can be easily applied within industry and the research community.

2.2.1 Malicious or Non-Malicious

Before we classify self-replicating programs we have to identify a common ground without compromise, where it is possible to single out a program as malicious or not. The term malicious is perceptional – one man's misery is another man's joy. For example, Stuxnet has hit some nuclear facilities, which might make it non-malicious (or beneficial) from the perspective of those who oppose nuclear power, but at the same time malicious in the eyes

[3]Although we briefly cover and define viruses, the work through the thesis is based on worms.

of those who support it. This ambiguity usually leads to frequent debates in the industry or the research community. Therefore, it seems that law is a reasonable referee to distinguish malicious acts to non-malicious ones. We therefore leave the distinction between malicious and non-malicious self-replicating programs – through this taxonomy – to the jurisdiction.

2.2.2 Viruses

Like worms viruses have always been portrayed as harmful and destructive, which has – in one way or another – impacted the way that they are defined and proposed. Two definitions, for instance, include:

> We define a computer ″virus″ as a self-replicating program that can ″infect″ other programs by modifying them or their environment such that a call to an ″infected″ program implies a call to a possibly evolved, and in most cases, functionally similar copy of the ″virus″ [85].

> A computer virus is a set of program instructions that attaches itself to a file, reproduces itself, and spreads to other files[77].

As it appears from the first definition it uses the word *infect* (inferring destruction) which is a prejudgment, while the other considers attachment to other programs behavior as a virus property. However, that might not be the case, since the virus attaches itself to other programs as a hiding technique – assuming a malicious intent – and not necessary a virus attribute. Even definitions that emphasize that a virus must be triggered by users [33] (human intervention) for execution, are not specific, since that is the case with most programs, not only viruses – even a worm in its first execution instance is triggered by a user. Therefore, a more accurate definition of a virus, would be:

Definition 2.1

A virus is a program that copies itself [90].

Thus any self-replicating program is a virus; regardless of weather it attaches itself to files, or any other actions.

2.2.2.1 Malicious Viruses

With no regard to intention, when a virus behaves in a way that violates a law issued by a legislative body, it can be considered malicious. Based on Def. 2.1 and Section 2.2.1, we therefore define a malicious virus as:
Definition 2.2
A malicious virus is a self-replicating program that – when released – breaches the laws issued by a legislative body.

Thus a virus that attaches itself to programs without user consent, exhaust computing resources, or any other harmful tasks that would lead to conviction before the court, would be considered as a malicious virus.

2.2.2.2 Benevolent Viruses

Bontchev discussed benevolent viruses in [14] and demonstrated the possibility of designing such programs. Although the author provided design guidelines, he did not give an explicit definition that would otherwise help identify benevolent viruses. Based on Def. 2.1 and Section 2.2.1 we therefore define benevolent viruses as:
Definition 2.3
A benevolent virus is a controlled self-replicating program that – when released – does not violate the laws issued by a legislative body.

Controlling virus propagation is essential, since it ensures that these self-replicating programs do not perform beyond authorized borders.

2.2.3 Computer Worms

Worms are a subclass of viruses (i.e. advanced viruses), that inherit the self-replicating property. Describing worms has been heavily debated and different definitions have been proposed in the literature [108]; however, in general they go around two main properties [104]: Self-replication and self-propagation using different communication mediums. Some of these definitions include:

> A worm is an independent program which, when run on a computer, will attempt to infect other computer systems [...] In this case the host program is the operating system of the computer, and the infected code is a stand-alone process or thread of execution running under the operating system [29].

> [a computer worm is] an independently replicating and autonomous infection agent capable of seeking out new host systems and infecting them via the network. [71]

> A computer worm is a program that self propagates across a network exploiting security or policy flaws in widely used services. [107]

Yet there is a more formal definition of computer worms, that can be found in R. T. Morris appeal (in 1991) which highlights the incident of releasing what has been known as the Morris worm (released 1988) [91]. The court defined a computer worm as:

> In the colorful argot of computers, a worm is a program that travels from one computer to another but does not attach itself to the operating system of the computer it infects. [47]

but as Nazario mentioned, this definition does not cover some worms that attach themselves to the operating system in order to hide their existence by using root kits or any other techniques [71]. Furthermore, the court definition along with some other definitions uses the word *infect* to describe the activity of injecting the worm's payload into the system – inferring a malicious intent – which make these phrases more appropriate to define malicious worms rather than worms in general. Following the existence of the area of defensive worms, using these words may no longer be adequate in worm definitions, since the injected payload might be for beneficial or non explicit malicious purposes (such as vaccination). Therefore, for the purpose of this thesis, a computer worm is defined as:

Definition 2.4
A computer worm is a self-replicating, self-propagating, and self-contained program that uses networking mechanisms to spread itself [81].

2.2.3.1 Malicious Computer Worms

Malicious worms – when released – breaches the laws issued by a legislative body. Even if these types of worms were released for good intentions, did not damage vulnerable systems or their networks; violating users privacy is enough to make these programs fall into the malicious category. Based on Def. 2.4 and Section 2.2.1 we define Malicious worms as:

Definition 2.5
A malicious computer worm is a self-replicating, self-propagating, and self-contained network program that – when released – breaches the laws issued by a legislative body.

Often malicious worms carry either a malicious payload or a payload without an explicit destructive intention in it, such as Slammer [63]. However, in general, such worms would not inject a beneficial payload into their targets. Damaging a network has been a common characteristic of malicious worms. Furthermore, malicious worms often propagate virally; as soon as they outbreak it is hard to stop or eliminate them. The faster and more viral the worm propagation is, the greater the impact and damage to the network. Examples of malicious worms include: Morris [91], Witty [65], Code red II [66], Blaster [11], and SQL Slammer [63].

2.2.3.2 Defensive Computer Worms

The research literature of defensive worms can not be described as mature. Researchers refer to these types of worms using different terms, including: *Beneficial worms, benign worms, good worms, benevolent worms, anti-worms, epidemic-style information dissemination, white worms, killer-worms, nematodes, helpful worms, good will mobile code, friendly worms, civilian worms, predators, counter-worms, or defensive worms.* Despite the term used, researchers agree that the main purpose of these worms is beneficial to the network. David Aitel defines a defensive worm as:

> A controlled worm that can be used for beneficial purposes [2].

It is worth mentioning that some researchers did not require a defensive worm to be controlled, yet it should have a beneficial payload [18]. However, previous implementations of worms with a beneficial use proved that uncontrolled (or viral) propagation might cause unexpected damage to the network. To avoid viral propagation, a main feature of defensive worms is the ability to control the way they propagate. Controlling worms, unfortunately, is a vague area in information security with little literature published except some techniques such as a query and response mechanism, something similar to DNS, which gives a green light for a defensive worm to target [2] or setting a time to live (or age) for the defensive worm. Other approaches might be utilizing topology information such as the Link Layer Discovery Protocol (LLDP), Content Addressable Memory (CAM) table, or Spanning Tree Protocol (STP) to define the path where these worms should travel [3, 4]. Based on Def. 2.4 and Section 2.2.1 we define defensive worms as:

Definition 2.6

A defensive computer worm is a controlled self-replicating, self-propagating, and self-contained network program that – when released – does not violate the laws *issued by a legislative body.*

From this definition, the worm can also be used for non security purposes, such as network topology mapping, traffic analysis, allocating undocumented nodes, and many other administrative tasks. However, in this thesis we consider the vulnerability mitigation side of this approach.

2.2.4 Botnets

Botnets are an extension of the worm class (i.e. advanced worms) where agents can interact with each other. In other words botnets inherit the properties of worms and add an interaction property. In the context of self-replicating code, we define botnets as:

Definition 2.7
A botnet is an interactive self-replicating, self-propagating, and self-contained network program.

Based on this definition when worm agents (or bots) have the capability to interact with each other, such a worm can also be described as a botnet. This description becomes necessary to avoid uncertainty and ambiguity when discussing worms and botnets. For example Conficker can be described either as a worm or to be more precise as a botnet (advanced worm), both descriptions can explain the program.

2.2.4.1 Malicious Botnets

As with malicious worms and viruses we define in a similar way a malicious botnet as:

Definition 2.8
A malicious botnet is an interactive self-replicating, self-propagating, and self-contained network program that – when released – breach the laws issued by a legislative body.

Examples of malicious botnets include: Storm [82], Conficker [23], and Stuxnet [70].

2.2.4.2 Defensive Botnets

As in defensive viruses and worms, defensive advanced worms (or botnets) should be controlled. We define defensive botnets as:

Definition 2.9

A controlled, interactive, self-replicating, self-propagating, and self-contained network program that – when released – does not violate the laws issued by a legislative body.

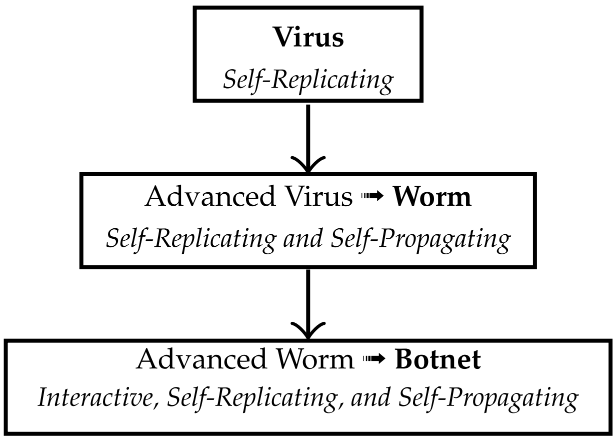

Virus
Self-Replicating
Advanced Virus ➟ **Worm**
Self-Replicating and Self-Propagating
Advanced Worm ➟ **Botnet**
Interactive, Self-Replicating, and Self-Propagating

Figure 2.2: Self-replicating Programs Classification.

2.3 Taxonomy in Practice

Based on this classification, viruses act as a superclass in which its attributes are inherited by worms [97], likewise botnets evolve to become a subclass of worms, see Fig. 2.2 for an illustration. This hierarchal view reflects the relationship between self-replicating programs and is a more constructive way to approach viruses, worms, and botnets. Which would help clear the confusion that sometimes takes place when researchers comment on incidents that involve self-replicating programs. It will also pave the way for considering the beneficial side of these programs. Table 2.2 lists self-replicating programs and classify them based on their properties.

☞ Program Name	**Virus** (Self-Replicating)	**Advanced Virus ➟ Worm** (Self-Propagating)	**Advanced Worm ➟ Botnet** (Interactive)
Bolzano	✔	✖	✖
Chiton	✔	✖	✖
Perenast	✔	✖	✖
Code Red I & II	✔	✔	✖
SQL Slammer	✔	✔	✖
Blaster	✔	✔	✖
Slapper	✔	✔	✔
Conficker	✔	✔	✔
Stuxnet	✔	✔	✔
Seawave	✔	✔	✔

Table 2.2: Virus, Worm, or Botnet?

2.4 General Attributes of Defensive Worms

There is a subtle distinction between a malicious worm and a defensive one; to avoid designing defensive worms that might engage in malicious activity, the industry needs some guidelines or a standard to ensure the quality of these solutions. It is not the payload that differentiates defensive from malicious, because worms with beneficial payload might still cause harm to the network [17]. Welchia, for instance, has been used as a countermeasure to the worm Blaster; it injected a beneficial payload, which downloaded a remediation patch from a central server. The problem was that Welchia overwhelmed the network with high bandwidth load, leading to high network disturbance [18] and flagging the worm as malicious – more details on Welchia is given later in Section 3.2. Also, while a beneficial payload distributed by a viral propagation technique is malicious, in the same way, a controlled propagation mechanism along with a malicious payload is malicious as well. Therefore, by combining the two factors, which are a non-malicious payload along with a controlled propagation technique, we might be able to see some difference between the two types of worms. Yet, that is not enough, a worm which is controlled and carries non-malicious payload, might still penetrate into unauthorized systems and thus should be treated as malicious. Therefore – inspired by Bontchev [14] – some design aspects can be considered before producing a defensive worm:

Payload The payload designed to be carried by the defensive worm should not be engaged in any malicious activity and should not exhaust computing resources.

Propagation The propagation of the defensive worm must be controlled to ensure that it stays within the scope of work and does not exhaust network resources.

Transparent The defensive worm should be easily recognized and perceived.

Uninstall The defensive worm should have the functionality to be completely removed or uninstalled.

Legal The defensive worm should not be designed to violate the laws issued by a legislative body.

Considering all these aspects before releasing a worm, would aid in deploying productive defensive worms. Still, it is a controversial area of discussion, arguments do exist between information security specialist regarding defensive worms. However, in general, a defensive worm should produce some degree of remediation and should not cause any harm to networks or violate any policies that it should, otherwise, adhere to.

2.5 Defensive Worms vs. Vulnerability Scanners

Defensive worms outperform traditional vulnerability scanners at least in three main aspects: Probing distance, ability to detect intermittent nodes and traversing the network regardless of network architecture. A shorter communication distance between a vulnerable node and the probing server is faster and less exposed to link failures compared to communicating to a remote IP address. Defensive worms install agents along their propagation path which hire each node to participate in the scanning activity and still maintain high scanning coverage even when an agent fails (i.e. no single point of failure). Scanners, however, scan different IP addresses remotely, thus adding more distance and more time for the scanners to communicate with their targets, and upon scanning server failure, the whole scope of vulnerable nodes is exposed. Furthermore, defensive worms keep probing their targets for vulnerabilities and detect any newly joining nodes or off-line nodes that become on-line. Yet, when vulnerability scanners conclude their assessment, newly joining or previously off-line nodes become exposed. Also, the self-discovering nature of defensive worms gives them the ability to spread around complex and large networks without high regard to the network architecture. Yet, scanners are usually required to be installed at each network segment for better performance [2].

2.6 Defensive Worms – Different Views

In this controversial topic, using worms for beneficial tasks often raise different views, some of which are hereby addressed.

2.6.1 Worms are not authorized to penetrate into their targets.

Worms can access network nodes using different techniques, not necessarily penetration. A worm can request permission from a user to access the system, use an assigned account, exploit a vulnerability, or other approaches. Releasing a worm should be within authorized scope (such as an enterprise network), should this not be the case, the worm becomes illegal as in any other solution.

2.6.2 Worms exhaust network bandwidth and hard to control and target.

Exhausting network bandwidth might be a common criteria of malicious worms, but is not a property of worms in general, and can be tackled by proposing new propagation algorithms which ensures that the worm traverses the network without disruption. It is the duty of the research community to design efficient controlling and targeting algorithms to further reduce the risks usually associated with self-replicating and self-propagating mechanisms – things appear hard until we become more aware of them.

2.6.3 Defensive worms do not stop zero-day attacks.

Defensive Worms are a tool to be used by security experts to prevent malicious attacks; tackling zero-day attacks is a design issue and depends on how worms are deployed – i.e. a behavioral based solution could detect a zero-day attack and releases a defensive worm to stop it.

2.6.4 Worms might leave a machine unstable and users might transfer the worm to other networks using their portable computing devices.

Like any solution, worms are tested before deployment to ensure they meet project goals. Controls can be proposed to prevent worm dissemination out of authorized scope, e.g. an agent does not propagate without a green light from a central node.

2.6.5 The risk when something goes wrong during worm operation is much higher than any other program.

This is a good reason to further research into worms and further understand them to minimize their potential risks. Like an airplane it is too risky if something goes wrong, but with comprehensive research, traveling now by airplanes is one of the safest ways of commuting.

2.6.6 When you add all necessary precautions, worms become so complex, that a patch management system would be enough.

Worms are not a replacement for patch management systems, they are tools that provide different features that can be utilized for different tasks. Worms are naturally simple as their self-exploring nature and ability to learn ease the need for detailed configurations.

2.6.7 Worms would propagate to non-enterprise nodes, such as plugged visitors portable devices.

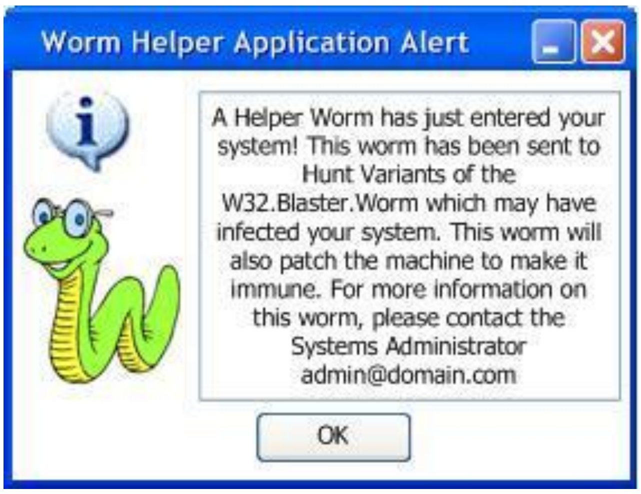
Visitor's devices should not access enterprise networks without authorization to ensure these devices do not host malicious code that might emigrate to the network. During the authorization process, devises can be excluded from the worm's scope (i.e. by IP or MAC). In general, giving non-enterprise devices full access to the network can not be considered a good practice. If we assume that a visitor node accesses the network without authorization, then since worms can access nodes using different techniques not necessary by exploitation such as by using an assigned domain account, then an access failure is expected when the defensive worm tries to access a visitor device. Upon access failure, network device information and location can be reported to the security team for action. These are design issues and is left for the security team to configure the worm according to its assigned mission.

2.6.8 There is no industrial need.

With current network threats, the industry and the government have gathered their efforts and issued calls for innovative solutions to keep consumer's ICT infrastructures protected from emerging and developing threats [61, 34]. Also, developing products that satisfy current industrial requirements is not a necessity for the research community; indeed very noticeable inventions – like airplanes, Global Positioning System, electronic computer, or Internet – have not initially evolved based on – at the time – market demands [41].

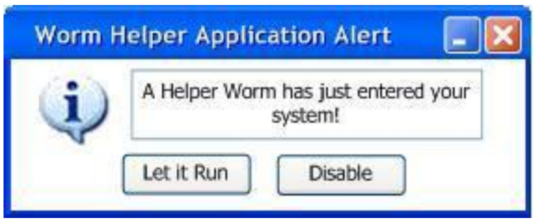
2.7 Wormophobia

Many researchers associate worms with malicious intent; indeed many observed worm outbreaks have caused great damage to the industry. This created a prejudgment that worms are always prejudicial to the network. This led industries to fear the use of self-propagating and self-replicating mechanisms as solutions to security or administrative problems. However, this fear factor [2, 45] can be eliminated or reduced to a level that would make defensive worms more acceptable commercially. Bellamy *et al.* investigated the reasons behind the fear of worms (Wormophobia) by conducting a survey to derive people's opinions and suggest steps to help escape from this fear circle [12]. They have selected a group of students of mixed age, level of education, and IT knowledge to ask them several questions with visual aids; examples of which are shown in Fig. 2.3, 2.4, and 2.5.


Worm Helper Application Alert

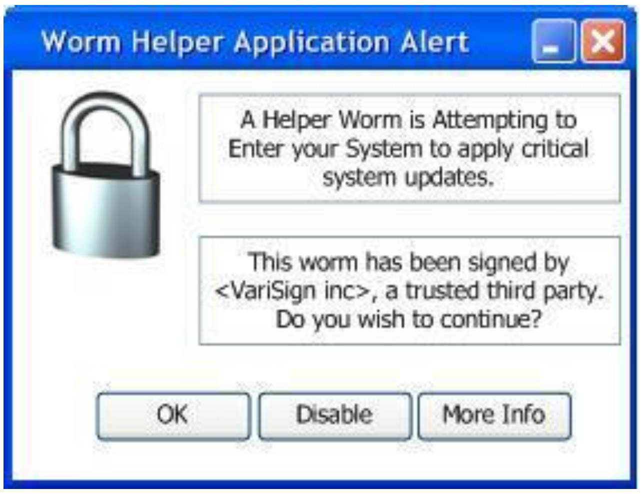
A Helper Worm has just entered your system! This worm has been sent to Hunt Variants of the W32.Blaster.Worm which may have infected your system. This worm will also patch the machine to make it immune. For more information on this worm, please contact the Systems Administrator admin@domain.com

Figure 2.3: "Would you allow a worm to run on your system if it stated its purpose and displayed contact information of who commissioned it?" [12]


Worm Helper Application Alert

A Helper Worm has just entered your system!

Figure 2.4: "Would you allow a worm to run on your system if you had the ability to disable it once it had entered your system?" [12]


Worm Helper Application Alert

A Helper Worm is Attempting to Enter your System to apply critical system updates.

This worm has been signed by <VariSign inc>, a trusted third party. Do you wish to continue?

Figure 2.5: "Would you allow a worm to run on your system if the worm had third party verification?" [12]

This study noticed that using visual aids, in addition to revealing more information about the mission of the worm (transparency) would help raise the level of acceptance towards defensive worms. The study also indicated that using the term *worm* in commercial use would make the user suspicious and uncomfortable and suggested using different terms to sell such products to wider customers.

Chapter 3

Defensive Worms – Related Work

3.1 Overview

Following the overview of defensive worms in Chapter 2; in this chapter we cover previous work that approached computer worms as a solution to network problems or as a defensive mechanism. Since the topic of defensive self-replicating and self-propagating network programs did not reach a reasonable level of maturity, up until now, we therefore cover researchers attempts from diverse resources.

3.2 Related Work

We divide this section into three subsections, with the first being an overview of research that adopts the use of worms for beneficial purposes. The second part highlights attempts carried out to release potential defensive worms; and the last focuses on general worm propagation techniques.

3.2.1 Defensive Worms in Literature

In the first research that used the term *worm*; the self-replicating and self-propagating network program was released for beneficial purposes. In 1979 Shoch and Hupp of Xerox Palo Alto Research Center deployed a worm that offered distributed computation within a network. The worm searches for idle machines and utilizes their free processor cycles to compute tasks; it allocates its targets by incrementing local host numbers. The worm does not exploit any vulnerabilities to gain access to an idle machine, but would rather request the machine to follow a certain procedure to transfer the worm. However, one morning, the worm went out of control and resulted in 100 machines malfunctioning. Fortunately, the authors included an emergency exit – when things go wrong – that enabled them to stop the worm. The control algorithm was later improved and another program was included to monitor, log, and control the size of the worm, leading the worm to run "flawlessly." Different applications were implemented including [87]:

- The Existential Worm: Used mainly to test the mechanism, doing nothing on the machine, but it could display a simple message and had the ability to self-destruct (or expire) after a randomly selected time.
- The Billboard Worm: Was used to display an image on the user's screen; the image is either included in the worm or is fetched from a central server.
- The Alarm Clock Worm: Uses an independent user program to allocate the worm and request a call to the user's phone. Like a wake up call calendar, the worm maintains a database of user's requests. When the time of the call is up, it connects to a terminal and calls the user.
- Multimachine Animation Using a Worm: A graphical engine that spans several machines and is controlled by a master node. Machines work in parallel on graphical frames before sending them back to the master node upon request. This application might be the first documented computer botnet – in the context of self-replication and self-propagating network programs.
- A Diagnostic Worm for the Ethernet: The worms here perform diagnostic operations, that require communication among machines and report results to the master node.

However, the experiment can be improved by utilizing topology information to control the worm propagation and enhance bandwidth utilization. It is not known how the proposed worms operate within large scale networks and if they can be easily recognized and perceived by users and network administrators. These worms also miss protective measures that ensures the confidentiality and integrity of its communications.

In the same direction, Toyoizumi *et al.* proposed a propagation model based on Lotka-Volterra (or predator prey) equations for a defensive worm that counters a malicious one. The defensive worm propagates by tracing the malicious one (e.g. by sniffing its traffic). When an infected node is detected by the defensive worm it penetrates the infected machine using the same attack vector as the malicious one and immunizes the system before propagating to a random number of randomly selected hosts, using the same method as the original malicious worm. Simulations with different scanning rates were performed on this model with their results being evaluated [102]. However, it is not advisable to use random scanning techniques, as it is hard to keep a worm using such propagation method operating within limits, let alone the excessive bandwidth generation usually associated with viral propagation techniques.

Wang *et al.* proposed a worm taxonomy to cover the aspect of worms used for beneficial purposes. They introduced a definition of a defensive worm which they have divided into two other worm types:

- SFworm: A defensive worm that penetrates into a susceptible machine using the same technique as the malicious one. It patches the vulnerable host to protect it from future infections.
- IFworm: A defensive worm that removes a malicious one and patches the infected system to prevent any potential breaches. It enters the infected machine using a backdoor installed by the malicious worm.

They have also proposed mathematical propagation models of these worms and simulated and analyzed them under different scenarios to counter malicious ones [104]. They used random scanning to locate targets and introduced a propagation technique that divides a scanning space into N subspaces, where the worm self-replicate to each subspace and so on until the worm kills itself when the scanning space becomes null. However, for more practical relevance, their proposal can be strengthened by providing technical details on how the worm divide the scanning space. The authors also do not highlight if the worm is authorized to penetrate into susceptible nodes, or provide any protective measure that prevents the worm from being hijacked by adversaries.

Similarly, Nicole *et al.* tried to measure the effectiveness of defensive worms by proposing four defense measures to counter malicious worms. These defensive types include:

- *Simple patch.* In this type, when a fixed number of nodes, scan (randomly) a susceptible node, it becomes patched.
- *Spreading patch worm.* Though similar to the Simple patch technique, this type of worm not only patches the susceptible node but also self-replicates before it starts scanning. Unlike the Simple patch, the patching hosts here continuously grow.
- *Nullifying defense worm.* In this technique, when the counter-worm scans an infected host it becomes suppressed, that is, the infection traffic is no longer effective.
- *Sniper worm.* In this technique, the counter-worm in the host listens for infection scans and upon detection it scans back the source of the infection (assuming the source IP address is not spoofed) and suppresses it.

Using discrete and continuos models, the authors evaluate these techniques as an active defense mechanism against malicious Internet worms [74]. In another work they also extended the Nullifying worm model to allow the worm to penetrate the infectious node, remove the infection, and self-propagate.

They have also compared some of these active techniques against passive measures, such as content filtering and address blacklisting [55]. However, as stated previously, using random scanning is not advisable; controlling the propagation of the worm either by utilizing topology information or any other factor, might be necessary to prevent the defensive worm from operating beyond limits or generating unnecessary bandwidth. Further enhancement would be providing protective controls to maintain the integrity and confidentiality of the defensive worm, in addition to, clarifying how the worm can be uninstalled from the network.

In addition, Castaneda *et al.* used defensive worms to counter malicious ones upon their outbreak into the network. Their active mechanism transforms a malicious worm into an anti-worm to provide immunization to infected or susceptible nodes. Through simulations they have evaluated their method against malicious worms such as CodeRed I, MSBlaster, and SQL Slammer. Their framework encompasses three stages: Detection, Analysis, and Generation. Upon detection the mechanism analyzes the malicious worm to detect which application is vulnerable and observes the changes occurred on the system before generating an anti-worm. The anti-wrom uses the same attack vector, to counter the malicious one. To spread the immunization payload, the mechanism uses four propagation approaches [18]:

- *Passive Anti-Worm.* In this technique the anti-worm listens for malicious scans and responds to the origin – disinfecting the source node.
- *Active Scanning Anti-Worm.* The anti-worm here uses random scanning to allocate its targets.
- *Active-Passive Hybrid Anti-Worm.* In this technique the anti-worm starts with random scanning before switching to passive propagation.
- *IDS-based Anti-Worm.* The anti-worm in this technique detects its tar-

gets based on intrusion detection devices scattered around the Internet. These devices capture suspected traffic and identify its source and destination for immunization.

However, this method can be improved by adding controls to prevent adversaries from abusing it, while making the defensive worm easily perceived and observed in the network for any monitoring or troubleshooting activity. The method also lacks a switching off technique or an uninstalling procedure.

Peikari, also investigated the possibility of using defensive worms to counter malicious ones. The author suggested different properties of an effective defensive worm, including:

- Providing a lasting immunity against the targeted infection.
- Consuming less resources than the targeted malicious worm.
- Distributing easily within the network.
- Should adhere to quality control procedures and eliminate its side effects as much as possible.
- The cost of developing a defensive worm should not be expensive for the organization and must reduce the potential damage caused by the malicious worm.

Different real world – event driven – simulations were conducted to test the defensive worm – using a nonhierarchical network topology. The simulations experienced a CodeRed outbreak and a defensive worm that uses CodeRed's propagation technique, however, to patch – rather than to infect – susceptible nodes. The defensive worm assumes the properties mentioned previously and is released before and after the malicious worm outbreaks, where an evaluation under different scenarios is conducted [24]. However, the legal aspect of defensive worms should always be highlighted and —

as noted earlier – adopting viral propagation techniques is too risky for the network. It is also not clear how to switch the defensive worm off or what protective measures are applied to ensure the integrity and the confidentiality of its communications.

For wireless sensor networks, Khayam *et al.* introduced and evaluated a Topology-Aware Worm Propagation Model to help in defending malicious worms and provide an effective vehicle to disseminate necessary information to secure the network. They divide the sensor network into rectangular segments (positions), where sensors (nodes) are uniformly distributed. A node in each segment can – at minimum – communicate with nodes inside the segment. For those nodes located at segment corners, the communication can expand to neighboring segments of up to r transmission meters. Each infected node self propagates to β fraction of its neighbors. In their model they have considered physical, MAC, network, and transport layer parameters [51]. However, it is necessary to address the legal aspects of releasing defensive worms in wireless networks and provide a method to uninstall and completely remove the worm from the network – in case worm deployment went out of plan.

Industry wise, Dave Aitel implemented a framework that can create a defensive worm (or a Nematode) that propagates based on simple incremental scanning. The automatically generating defensive worms framework converts an exploit into a payload to be used by the worm. This is probably the first attempt from industry to consider worms for beneficial purposes. However, no further development or marketing attempts about the framework have been observed [2].

Based on [102], Gupta *et al.* proposed three propagation models of defensive worms as follows:

- Persistent predators; which adds a delay before the defensive worm dies.

- Immunizing predators; which access vulnerable systems disinfect and install patches.
- Seeking predators; which has the ability to follow the propagation path of the malicious worm.

They have run different simulations on a non-hierarchal fully connected network testing their propagation techniques to counter malicious worms followed by results analysis and evaluation. They have also addressed some central patching drawbacks, such as the possibility of bottlenecks occurring during patch distribution in large scale networks and the vulnerability of denial of service attacks usually associated with server-centric approaches. They also suggested few methods to access a user machine without exploitation and indicated that based on their experiments, defensive worms can be a promising alternative for centralized patching [35]. However, the proposed worm propagates randomly making it hard to control and there is no suggested controls to protect the defensive worm from malicious users. Further improvement can also be making the worm easily perceived and observed in the network for better monitoring and troubleshooting.

In the same direction, Wu et al. adopted the approach of containing a malicious worm by releasing a defensive one; they have divided defensive worms into three types [109]:

- Patching, which only installs a patch on vulnerable nodes.
- Predator, which removes the malicious worm from infected hosts (with the ability to install a patch).
- Composition, which removes the malicious worm from infected nodes and patch the vulnerable system.

The approach can be enhanced by using topology information to control the way these worms propagate and adding protective measures to prevent any misuse of the defensive worm. The approach also lacks a safe exit in case things went wrong or a clear uninstalling procedure.

In another strategy, Liu *et al.* proposed an epidemic model of a defensive worm that spreads within a directed graph such as there is no repeated edges and the in degree of each node is one while the out degree of each node is the same. In their Balanced Tree based Propagation strategy model, the worm infects the node, then moves forward to its descendants before deleting itself on the infectious node. Using Matlab they have run different simulations of defensive vs. malicious random scanning worm followed by a performance evaluation of their strategy [56]. However, for practical relevance, the authors should provide technical details on how the worm define its propagation path. Further improvements can be adding measures to protect the worm's line of communication from malicious abuse and providing a technique to remove or uninstall it, in addition to, addressing the legal background behind releasing the worm.

Meanwhile, Tanachaiwiwat *et. al* studied the defensive worm vs malicious worm technique and identified three types of worm interactions:

- *One-Sided.* One worm counters another worm in a way that mimics the predator/prey technique.
- *Two-Sided.* Two worms try to finish each other. Predator against another predator type of interaction.
- *Indirect.* Two worms exist in the same network without trying to overcome each other.

They model these interactions and evaluate their performance through simulations taking into consideration two factors: Scan rate ratio and initial infected host ratio [98]. They also extend their work by proposing a model of an aggressive one-sided interaction considering network delay factors such as packet size, latency, queuing algorithm, and bandwidth. They also provide a metric to measure the damage caused to the network by the malicious worm after the release of a defensive one, in addition to identifying some factors that affect malicious worm containment procedure [99]. On the same path as Castaneda *et al.* the same authors proposed a model of a defensive worm encountering a malicious one (aggressive one-sided worm interactions), however, under an encounter-based network which is a "frequently-disconnected wireless ad-hoc networks requiring close proximity of neighbors." In their work they concentrated on mobile node characteristics such as cooperation between nodes, immunization, and intermitted node behavior [100]. However, it is not clear in which legal ground the authors expect the worms to be released and weather the worms are easily perceived and observed in the network. The work can be further extended by considering controlled worm propagation techniques and securing the mechanism from possible adversaries misuse.

Based on the two-factor model [116] Zhou *et al.* put forward a mathematical propagation model of a passive worm and analyzed the possibility of using passive worms for defensive purposes. They also identified the initial value of passive worms, the scanning rate, the removal rate, and time delays as factors that affect the performance of these worms [115]. Following the classification of worms in [18] and the classification proposed in [109] the authors further classify Hybrid defensive worms into three types:

- Patching-hybrid defensive worm: The defensive worm combines the attributes of both a Patching and a passive defensive worms.

- Predator-hybrid defensive worm: The defensive worm combines the attributes of both a Predator and Passive defensive worms.

- Compositive-hybrid defensive worm: The defensive worm combines the attributes of both a Compositive and Passive defensive worms.

They derive models of each type taking into consideration time delays and run simulations and evaluate their performances [114]. The work, however, disregards the legal issues regarding releasing such defensive worms and does not provide an uninstalling or removing method after worm deployment. The work can be further improved by considering controlled propagation techniques and adding controls to ensure the defensive worm is not utilized by malicious users.

Different theoretical epidemic-style probing strategies were proposed by Vojnović *et al.* based on random probing. They identify the minimum number of probes required to reach targeted number of nodes, assuming the distribution of susceptible nodes among the network is known. They used a variable probing rate that changes based on time (dynamic strategy) or a fixed probing rate (static strategy) in an attempt to recognize the most suitable performance to disseminate information using random probing. For dissemination where host distribution is not known, each infected host initiates probing based on information received from the original infecting node. In this strategy, the worm switches between local subnet probing and global address space random probing, based on an observed number of failed probing attempts (K-Fail). Another probing strategy picks a host randomly from within a randomly selected subnet in a list stored in the infectious host, or picks a host randomly from the entire address space (K-CANDSET). They also evaluate the parameters that affects the performance of their proposed strategies based on different Internet datasets [103]. However, the work does not provide technical details on how these propagation techniques can be implemented which is necessary for any practical relevance. Also it is necessary to address the legal issues regarding self-replicating propagation techniques and how these techniques can be protected from malicious abuse.

From a human rights perspective, Aycock tried to consider defensive worms to measure to what extent Internet censorship is applied in China.

He proposed ideas and techniques to be utilized by, what he named a *human rights worm* [10]. No results about the worm performance has been reported.

Inspired by worm vaccinations, Wang *et al.* proposed a Susceptabe-Exposed-Infected-Quarantined-Vaccinated (SEIQV) epidemic model that makes use of vaccination and dynamic quarantine techniques to reduce the number of infected hosts and further contain the malicious worm. They have also studied the impact of different parameters on their model, and run simulations to evaluate its capabilities [105]. For further improvements, the work can address the legal ground behind releasing the worm and provide protective measures to keep the mechanism safe from adversaries.

Berbar *et al.* considered using beneficial worms to test distributed systems and verify the fault tolerance capabilities of such scattered software. The worm closely monitors each entity of the distributed system then reports error messages to a specific worm node for administrators to inspect [13]. However, it is not clear if the worms are easily perceived and recognized within the network and it is necessary to add an uninstalling functionality in case these worms performed beyond expectations. Also work can be further improved by addressing how the worm would perform in large scale networks, in addition to, adding protective measure to keep the worm operating away from malicious interference.

While, Yao *et al.* proposed a system that combines honeynets with anomaly detectors to recognize and prevent malicious intrusions. Upon attack detection, each honeypot will release a peer to peer defensive worm to encounter the malicious one. They have provided a *P2P-based Benign Worm Model* and run simulations to assist and evaluate their work [111]. However, the legal ground in which the defensive worm are released is not clear and protecting worm communications is not highlighted to prevent adversaries from hijacking the mechanism.

In the same direction, Toutonji *et al.*, proposed a Passive Worm Dynamic Quarantine (PWDQ) model, which describes a method to stop malicious worms by recovering infected hosts either by dynamic quarantine or passive benign worms techniques [101]. However, the approach can be further enhanced by providing a procedure to remove the defensive worm when released, highlighting the legal ground on which the worm can be deployed, clarifying how the worm can b easily observed in the network, and securing the mechanism's communications to block any malicious interventions.

Yong, improved on a previous vaccination structure that uses worms to counter malicious ones. His modified structure does not require the worm to download the payload from a central server and instead adds a "Decoding code" and a "Resuming and Execution exe" phases to avoid the requirement of central servers, download connections, and download code [112]. Further improvements might be adding an entity to the structure responsible to stop or remove the counter-attack worm, to provide a safe exit in case the worm went out of control.

Moreover, Nie *et al.* viewed peer to peer defensive worms as an adequate solution to prevent malicious P2P worms. When two peers exchange files the node with a higher patch version transfer the security patch to the lower patch version node. They have provided a model for their work and run several simulations to evaluate and examine their approach [75]. However, it is not clear how the mechanism would perform in large scale networks or how it can be uninstalled or removed from the network.

Al-Salloum *et al.* introduced a defensive worm that utilizes information within the Link Layer to reconstruct topology information discovered through the *Link Layer Discovery Protocol* in order to detect neighboring vulnerable nodes and propagate gradually until total coverage of the enterprise network is reached [3]. While in another work they proposed what seems to be the first computer worm that utilizes layer two of the OSI model as its main propagation medium. They introduced a defensive worm that utilizes topology information such as *Content-Addressable Memory* (CAM) tables and *Spanning Tree Protocol* (STP) stored in switches. In this approach, the vulnerability mitigation mechanism propagates through traversing switches whilst probing vulnerable hosts until the network is covered [4, 5].

3.2.2 Implemented Defensive Worms

Few attempts were taken to deploy potential defensive worms in real world networks. Most worms were released to confront and eliminate other malicious worms, however, these attempts were not successful as they have violated different laws. These worms, include:

Welchia. The Blaster worm variant [46, 30], was released to counter the spreading of Blaster. Welchia, exploited the same vulnerability at the same TCP port as Blaster to propagate. The vulnerability has been addressed by Microsoft Security Bulletin MS03-026 as a buffer overflow in Microsoft Remote Procedural Call (RPC) service. Welchia, immunized a susceptible system by exploiting the vulnerability and downloading the MS03-026 patch then rebooting. However, the worm did not succeed in accomplishing its goal and instead has caused more damage to the network. That is due to two reasons: first, Welchia generated massive bandwidth by downloading patches from the vendor server (windowsupdate.com); secondly, Blaster – in addition to the traffic generated by Welchia – launched a denial of service attack at windowsupdate.com. Furthermore, one variant of Welchia tried to surpass the propagation speed of Blaster by increasing the propagating threads to 300 (originally was 50) and the worm used an unrestricted ICMP scanner with a short timeout, which made it become more unstable. Such techniques to increase Welchia's speed over Blaster have backfired by generating more network traffic [18, 104].

Different views have been expressed on Welchia, especially in regard to its nature, weather it was good or bad? Even the intention behind releasing the worm has been questioned, as it has been reported that some Welchia variants have installed an unrestricted backdoor at their targets. Some other views looked at Welchia as another malicious worm that tries to compete with the Blaster worm [18]. Based on our definition of malicious worms 2.5, Welchia can be considered malicious, due to the damage it caused to the Internet and its users, which violates laws issued by legislative bodies.

CRClean is a Code Red II variant, which exploits a buffer overflow vulnerability in the Index Server plug-in in Microsoft IIS Server as announced by Microsoft Security Bulletin MS01-033 [18]. CRClean has been designed and presented by a German coder named Markus Kern [43]. CRClean has used an interesting way to spread, it only spreads to systems that have attempted to attack it. This technique is sometimes referred to as *passive propagation*. The technique would normally reduce the network traffic usually induced by worms that use active scanning to propagate. Another technique utilized by CRClean was the intercepting of Code Reds malicious traffic, which blocks any future infections of targets [18].

CRClean works by silently running on a system, waiting and listening for Code Reds attacks. When CRClean intercepts an attack attempt from a system infected by Code Red, it launches a counter attack to remove Code Red and installs CRClean at the system that has launched the attack [57]. Furthermore, since CRClean is memory resident, it removes itself from the system, once the system is shutdown (if the date is November, 2001 or Later). Furthermore, for the worm to be detected and identified, it adds a unique signature to server logs in which it has penetrated. Although, CRClean has introduced some interesting techniques, it has not actually been released on the Internet [18]. Even if it was released, it would most probably break into users machines without authorization, which is – at least – a violation of the privacy law, which puts it in the malicious category.

Code-Green. Also written by a German programmer, who goes by the name Herbert HexXer. Code-Green is designed to combat the Code Red II worm. Because Microsoft IIS server can be exploited more than once, Code-Green is able to attack Code Red infected systems [97]. Code-Green takes several steps to eliminate Code Red. Once running on the machine, Code-

Green writes a signature into the system memory identical to the way Code Red's writes its own signature. This signature is called an *atom*, which ensures that Code-Red will not re-infect the system again, as it appears to be infected – while it is not. Next, Code-Green attempts to remove the backdoor (root.exe) installed by Code-Red. This backdoor provides a remote attacker with full control on the IIS server. Also, Code-Green will disable the virtual mapping previously created by Code Red II, which exposes drive D and C to remote attackers. Then, Code-Green tries to determine the language of the system to pick the correct patch. Subsequently, it downloads and installs the Microsoft patch MS01-033 to fix the vulnerability already utilized by Code Red II. Furthermore, Code-Green scans for systems that are still infected by Code-Red II and patch them as well. Code-Green is memory resident, which means that once the system is rebooted Code-Green is removed [57]. Yet, Code-Green writes a unique signature to the logs of the targeted systems, indicating it has been there [43]. Although, Code-Green was successfully tested on German language systems, it has not been released on the Internet [57, 18].

Cheese. Beneficial Worms are not meant only for Microsoft Windows systems, but also for other platforms including Linux. Cheese was written and released to overcome and clean the damage caused by the Linux targeted malicious worm Lion (or 1i0n). On UDP port 53 (DNS), the Lion worm was able to exploit a buffer-overflow vulnerability in the Transaction Signature component of the BIND 8.1 server [71, 18]. Lion installed a backdoor listening on TCP port 10008, which binds to a command shell when connected. Cheese scans for hosts that have the TCP port 10008 open and in listening state, and then it connects to the port in order to propagate [71].

The worm scans IP addresses that belong to net blocks 193-218.1-254/16 randomly. When Cheese connects to its targets (port 10008) it gets bound to a command shell where it can issue commands on the infected system. The worm initiates a series of commands to load itself and remove any instances of the malicious worm Lion, in addition to disabling the backdoor service from inetd [71, 18]. Although Cheese was meant to fix and clean its targets, there were some observed disruptions and confusion on the network; that is due to the immaturity of this technique [71].

Unlike Code Green, Cheese has been released on the Internet; however, it was released in a small scale, and there was not much known effect on the network [18]. It worth mentioning that Cheese breaches an important feature of defensive worms, in which it propagates randomly, making the worm spread out of control, let alone accessing users systems without consent, leading the worm to be classified as malicious.

Anti-Santy. This worm was released to counter the Santy malicious worm, which infects websites that uses phpBB based web forums [49]. Anti-Santy, uses the same propagation method as Santy, which is by issuing specially crafted search requests to search engines, like google, to detect Internet forums that run a vulnerable version of phpBB software. The worm then penetrates these sites and downloads and installs a patch to recover the vulnerability. Anti-Santy also defaces websites it breaks into with a messages such as: *viewtopic.php secured by Anti-Santy-Worm V4. Your site is a bit safer, but upgrade to* $\geq$ *2.0.11* [58]. The worm can not be described as beneficial as it generates high traffic towards infected websites, slowing them down. Moreover, the worm defaces webpages without the owner's permission, which violates laws issued by legislative bodies, making Anti-Santy fit more in the malicious category.

3.2.3 Worm Propagation Techniques

Computer worms differentiate based on the way they propagate. Worms have adapted different techniques to locate vulnerabilities and propagate to cover as many targets as possible. In this section we cover some of these techniques, where most of them were utilized by malicious worms and are not suitable to be used for beneficial purposes unless accompanied by propagation control measures. Worms like Code Red I and Slammer have used random scanning techniques to find their targets, while even the Morris worm (which utilized local subnet topology information) used a more intelligent way of propagation [92]. The propagation speed increases when the worm incorporates information to locate all vulnerable hosts, such as in the Flash worms [93].

A "hit-list" worm would incorporate information about a number of vulnerable hosts where then it will switch to random scanning after scanning all the hosts in the hit-list. This was proposed by Staniford *et al.*, with further targeting enhancements, suggested e.g. by Fan and Xiang [26].

Worms like SQLsnake, use a built-in list of numbers that will be used later to generate network addresses to probe for vulnerabilities, these numbers are generated according to network space that most probably contain vulnerable targets [71]. Zou *et al.* proposed a *routing worm* which scans based on the information provided by the Border Gateway Protocol to reduce scanning space [118]. They also introduced a *divide-and-conquer scan worm* which uses the divide and conquer approach to propagate. When the target is infected it passes half of the scanning space to the target and continues scanning the other half of its original space [117].

Code Red II used different scanning techniques where hosts closer to the infected target are scanned with higher probability than those farther away [66]. This is a scanning technique referred to as *Island Hopping*, as the network is viewed as a collection of islands, where an island receives specific attention before hopping to another island [71].

Vojnović *et al.* identified optimal static and dynamic propagating strategies. These proposed strategies minimize the total number of sampling to reach a target fraction [103]. In one strategy, a worm infects a randomly selected host then tries to spread on the same subnet, as long as there are many vulnerable hosts. If not, the worm shift's to another subnet using random scanning [88].

Markus Kern has introduced CRclean, a worm that spreads passively by listening to Code Red I scanning attempts. When Code Red I scans a host that already has CRclean installed, CRclean will respond to the scanning activity by reinfecting the scanning source host and removing the malicious worm to provide remediation and containment of Code Red I malicious spread [50]. Blacklists were used by Conficker, the list contained entities that might provide remedies and containment actions towards malicious code, such as anti-virus sites and Microsoft, the worm will not try to scan IPs in the list to avoid detection [53].

3.3 Summary

In this chapter we described attempts that involved utilizing worms for defensive or beneficial purposes. These attempts were covered based on published literature or actual attempts of releasing potential defensive worms. We have also highlighted the propagation techniques that worms usually use to disseminate around the network, as worms mainly differentiate based on the way they propagate. This chapter tries to set – at least – a semi comprehensive resource for previous work that involves defensive worms to aid interested researchers in their efforts to explore this topic of research.

Chapter 4

A Vulnerability Mitigation Worm – Seawave I

4.1 Overview

We have defined and provided a taxonomy and an overview of defensive worms and the challenges the industry face in preventing or reducing the severity of network vulnerabilities in Chapter 2; followed by a summary of previous work that involved worms for beneficial purposes and worm propagation techniques at Chapter 3. In this Chapter we present a novel controlled, interactive, self-replicating, self-propagating, and self-contained vulnerability mitigation worm named *Seawave*[1]. The mechanism probes for vulnerabilities within an enterprise network by plotting agents during its gradual propagation. The defensive worm utilizes layer two topology information collected from network switches to achieve minimum bandwidth usage and maximize network coverage. To the best of our knowledge Seawave is the first computer worm to utilize the data link layer (of the OSI model) as its main propagation medium.

4.2 Network Topology Model and Simulation Environment

In the design of the simulation environment, we have taken into account where the mechanism will most probably and most beneficially be deployed. The networks were designed in a hierarchical manner with a varying num-

[1]Named after the observed wavy propagation curve of the mechanism, which mimics the waves of a sea – see Fig. 9.4 of Chapter 9.

ber of Local Area Networks (LANs) connected to each other through a backbone. These hierarchical topologies mimic, to a large extent, enterprise networks where the mechanism experiences disparate bandwidth capacity within the backbone and main network links as well as the considerably higher bandwidth available within switches.

For modeling purposes, we divide network nodes into three types: Hosts, switches, and routers. Fig. 4.1 illustrates an example of a 1000 node network. For more accuracy in the results, different numbers and layouts of hierarchical networks have been designed for simulations. Network nodes have been chosen randomly and linked to randomly selected switches under a specified probability; switches are then connected randomly based on a specified probability to form an acyclic local topology. After forming the LAN, a switch is then selected randomly to be linked to a router located in the backbone to form a LAN connected to the backbone; this process repeats until the whole enterprise network is completed.

The Drop Tail queue management algorithm [21] was used as a queuing algorithm and for performance measurements, we have – without loss of generality – assumed duplex network links with 100Mbps bandwidth; the simulations in Section 4.7 have used different networks with a node number varying between 100 and 8000.

For our simulations we have assumed the following:

- The network implements STP and is loop-free.
- Simple Network Management Protocol (SNMP) [38] is supported.
- Vulnerability discovery packets have a length of 900 bytes.
- CAM/port status is maintained by switches.

We have generated 17 hierarchical networks consisting of different numbers of LANs, routers, switches, and nodes.

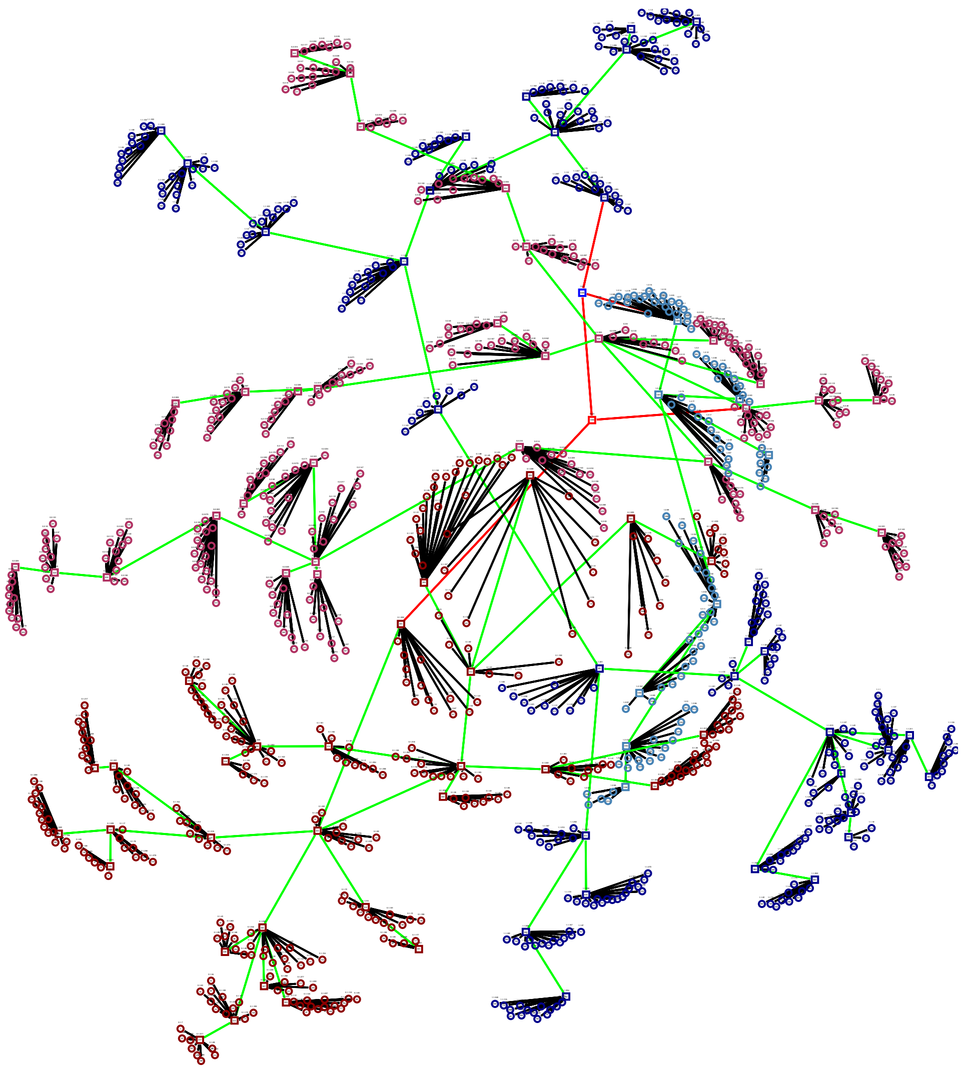
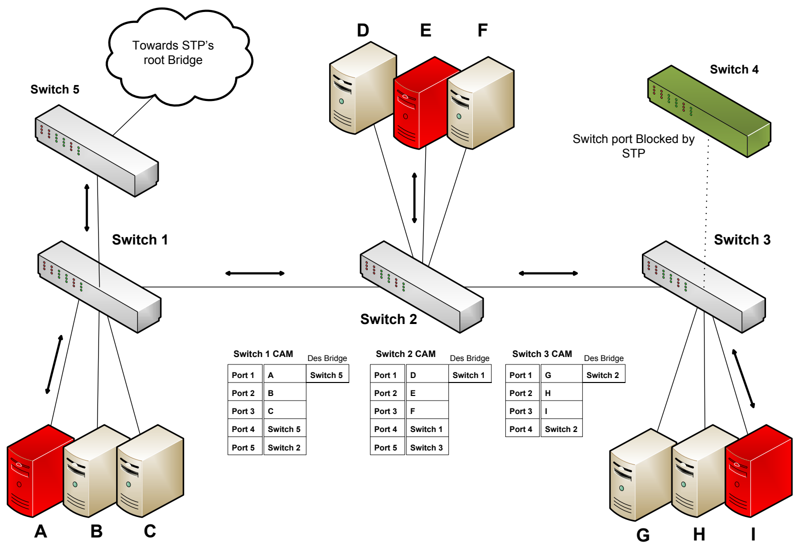


Figure 4.1: Switched network that implements Spanning Tree Protocol

4.3 Seawave I

Seawave is a *controlled, interactive, self-replicating, self-propagating, and self-contained vulnerability mitigation worm.* It utilizes network topology knowledge to propagate in a constructive manner that would not impact network resources. It uses the Spanning Tree Protocol (STP) information to identify switches and then reads Content Addressable Memory (CAM) tables stored in switches to identify nodes and directly connected switches [1] . Note, that topology information is read step-by-step (and not as a whole) as the



Switch 1 CAM

		Des Bridge
Port 1	A	Switch 5
Port 2	B	
Port 3	C	
Port 4	Switch 5	
Port 5	Switch 2	

Switch 2 CAM

		Des Bridge
Port 1	D	Switch 1
Port 2	E	
Port 3	F	
Port 4	Switch 1	
Port 5	Switch 3	

Switch 3 CAM

		Des Bridge
Port 1	G	Switch 2
Port 2	H	
Port 3	I	
Port 4	Switch 2	

Figure 4.2: Seawave I

defensive worm propagates gradually. Based on this, paths are constructed dynamically and on a switch-by-switch basis, updating STP and CAM information on each step to discover directly connected hosts and switches.

To illustrate the propagation process, assume that in Fig. 4.2 Seawave was installed on node *A* where it reads STP and CAM information from *Switch 1* and performs vulnerability discovery on Stations *B*, *C* and simultaneously reads CAM tables of neighboring switches, in this case *Switch 2* and *5*. The Seawave agent then process the CAM tables and picks a station at each neighboring switch to self-replicate. The agent picked station *E* to propagate to *Switch 2* then the same procedure repeats until the Spanning Tree path is covered.

For concurrent propagation, Seawave assumes all nodes are vulnerable. In Fig. 4.2 nodes *A*, *E*, and *I* will act as a vulnerability discovery server for the nodes directly connected to the switch. This will lower the transmission distance, would keep any bandwidth generated within this process contained within leaf edges without impacting the heavily used network links, and would allow the defensive worm to detect un-propped nodes quickly. For example if a node was off-line during the process of vulnerability discovery, then when the node is back on-line the vulnerability discovery server of the switch will try to detect the new node and thereafter probe it; identifying intermittently active hosts is detailed in Chapter 5.

The resources for the agent to start detecting vulnerabilities should not require a large amount of CPU or memory to probe few host nodes connected to one switch. This allows the agent to share station resources with minimum overhead. Otherwise, alternatives can be considered, such as probing when the server node is on idle status – bound to the severity of the vulnerability.

4.4 Propagation Algorithm

For Seawave to propagate around the network it has to visit each switch attached to the spanning tree. We consider the whole network as *N*, where $N = \{LAN_0, LAN_1, \dots LAN_n\}$. Each subnet consist of one router and several switches and hosts, which represents one domain (i.e. for the purposes of this thesis we do not consider VLAN configurations; these can be handled analogously), and each switch in the subnet exists as part of the spanning tree path of that particular subnet.

We denote STP_i as the spanning tree information stored in switch *i*. $STP_{i\{BridgeID\}}$ denotes the Bridge ID of switch *i* and $STP_{i\{DesBridge\}}$ denotes the designate Bridge of switch *i*, which is the next bridge in the ST path and we denote S_{ij} as interface *j* at switch *i*. For the switch to decide where to forward packets, it refers to its CAM table (or MAC forwarding address table). When the switch receives a packet it compares the destination MAC with those entries in the CAM table. If there is a match, the packet will be forwarded to the designated port, otherwise it is sent to all ports [86].

We denote the CAM table of switch *i* at port *j* as C_{ij} and the MAC address of switch *i* as S_{iMAC}. In order for an agent to start spreading between switches, it has to be able to detect a direct connection between two switches. A direct connection between two switches is when two switches – which exist in the spanning tree path – are connected with no other elements between them. This will enable the agent-based mechanism to propagate gradually and avoid redundant probes.

We therefore first state a lemma that builds a sufficient base for a switch directly connected to another switch assuming that the IEEE specification of the Spanning Tree Protocol [1] is honored:

Lemma 4.1

Interface S_{ij} and S_{kl} are directly connected to each other if and only if $STP_{i\{Des.Bridge\}} = STP_{k\{BridgeID\}}$ when $S_{ij} \rightarrow S_{kl}$ or $STP_{k\{Dec.Bridge\}} = STP_{i\{BridgeID\}}$ when $S_{kl} \rightarrow S_{ij}$

It is not straightforward to detect directly connected switches. Few approaches exist to detect direct connections. One of them proposed by Breitbart *et al.* [15], states that two switches interfaces S_{ij} and S_{kl} are directly connected if and only if $C_{ij} \cup C_{kl} = u$ and $C_{ij} \cap C_{kl} = \Phi$. u here denotes all MAC addresses of routers and switches in a subnet S. However, the assumption that a CAM table at a given interface does contain all the MAC addresses that can be received from the same interface is not satisfied in all real-world networks [15] owing to aging properties of CAM tables and communication demand. When a network element becomes idle on that interface, the source MAC address might be removed from the table affecting the above assumption. Furthermore, an element might not need to perform any network activity that passes through that specific switch interface, resulting in the MAC address of the element not existing in the CAM table at all.

Another approach is proposed by Stott [95] where the spanning tree information is read using SNMP from all switches and routers to construct the network topology. However, this approach will generate bandwidth according to the number of switches and routers which is large in corporate networks; in addition, the topology might go through different changes in the process of collecting STP information.

In Seawave, the agent, however, tries to combine both CAM table information and Spanning Tree information to detect a direct link between two switches and therefore propagate switch by switch until the LAN is covered. In order to detect directly connected switches, we assume that each CAM table of each switch holds at least the MAC address(es) of the directly connected switch(es). Switches exchange Bridge Protocol Data Unit packets [1] that hold spanning tree information quite often (every two seconds by default) [86] enabling this assumption to most likely be reflected in real world networks.

Under the assumption that the network implements STP according to the standard IEEE 802.1D [1] with default settings we now introduce the following Lemma:

Lemma 4.2

If S_{ij} is directly connected to S_{kl} then $\{S_{iMAC}, S_{kMAC}\} \subseteq (C_{ij} \cup C_{kl})$ and $C_{ij} \cap C_{kl} = \Phi$

PROOF Assume that S_{iMAC} does not exist in C_{kl} and S_{kMAC} does not exist in C_{ij}, therefore, $\{S_{iMAC}, S_{kMAC}\} \nsubseteq (C_{ij} \cup C_{kl})$. However, according to lemma 4.1 $STP_{i\{Des.Bridge\}} = STP_{k\{BridgeID\}}$ or $STP_{k\{Dec.Bridge\}} = STP_{i\{BridgeID\}}$ and according to the default STP settings a switch would emit BPDU frames once every 2 seconds, which is much less than the default CAM table aging time (5 minutes) [86]. Thus, there exist bidirectional STP traffic between the directly connected switches S_i and S_k. As a consequence, C_{kl} contains S_{iMAC} and C_{ij} contains S_{kMAC} which is a contradiction ■

The ability of the agents to propagate through the network then follows from lemma 4.1 and 4.2.

We denote the hosts connected to the directly connected switch as $H(C_{s_0})$ where C_{s_0} is the CAM table of switch s_0. And $S(C_{s_0}) = \{s_1, s_2, ..s_n\}$ as switches listed in the CAM table C of switch s_0; while $(C \cup STP)_{s_0}$ as the CAM table and STP information of switch s_0. We denote the function of fetching data through SNMP as $SNMP$ and agent self-replicating as SR. And the function of vulnerability detection and selecting a host from a list as VD and $select$, respectively. The propagation algorithm of Seawave, then follows:

1. Install Agent at the starting host at corporate subnet.
2. Agent starts self-replicating to all subnets according to predefined hosts in each subnet: $SR(LAN_1 \cup LAN_2.. \cup LAN_n)$
3. In each subnet the agent reads STP and CAM table information from the directly connected switch: $SNMP\{(C \cup STP)_{s_0}\}$
4. The agent extracts directly connected hosts from the CAM table then performs vulnerability detection to these hosts: $VD(H(C_{s_0}) - \{H_{agent}\})$
5. Based on lemma 4.2, agent reads STP and CAM table information from switches listed in the CAM table of the directly connected switch ($S(C_{s_0})$): $SNMP\{(C \cup STP)_{s_1} \cup (C \cup STP)_{s_2} .. \cup (C \cup STP)_{s_n}\}$ and then detects directly connected switches based on lemma 4.1[2].
6. The agent picks a host in each neighbor switch and self-replicate to propagate on the same subnet: $SR(select(H(C_{s_1})) \cup select(H(C_{s_2})).. \cup select(H(C_{s_n})))$
7. Go to step 3 until all neighboring switches in the STP path are visited.

Seawave depends on the information retrieved from CAM tables and STP that are stored at each switch. Fortunately, it is not necessary to rely on proprietary protocols, as it is possible to adopt widely supported standards, such as SNMP management information base (MIB) objects to retrieve this

[2]For the directly connected switch towards STP root, the agent can consider S_{kMAC} address instead of requesting $STP_{k\{BridgeID\}}$ – assuming that the Bridge ID consist of the switch MAC.

information, giving Seawave more flexibility and portability over different networks with different types of switches.

4.5 Vulnerability Detection

In our simulations, we have assumed that one packet is enough to detect a vulnerability and install an agent; which mimics – to some extent – SQL Slammer which uses a single packet to propagate [63]. Of course, one packet is not enough to contain a payload that performs complex tasks (or deal with multiple vulnerabilities), however, for such tasks larger payloads can be used.

Different approaches exist to detect a vulnerability, one of them is by exploiting. The agent will try to probe a potential vulnerable machine by sending a packet with the necessary payload to achieve three tasks.

- First, exploit the vulnerability to gain the necessary privilege to apply temporally remediation.
- Second, apply vulnerability remediation to eliminate the security exposure of the vulnerable machine.
- Third, trigger the agent for further propagation to cover other vulnerable nodes.

Vulnerability remediation is temporary and clearly is not substitute for a code-level patch and can range of different techniques, such as disabling a port that can be used by a malicious user to compromise a machine, or installing a wrapper script that will act as a packet filter between the vulnerable application and the network, or even uninstalling the vulnerable application. Whatever remediation is used by the network security team, it must have the required privilege to assure successful deployment.

Seawave, should be deployed carefully as its payload might be exposed to network users with malicious intention; but the mechanism will most likely be triggered by the enterprise security team in response to a critical vulnerability with an already publicly available exploit – more risks and threats are addressed in Chapter 5. If exploitation was successful, the node is vulnerable, otherwise, the node most likely is not. This approach would eliminate the amount of false positives, usually reported by vulnerability discovery applications [83]. However, the exploitation procedure must be performed carefully as it can lead to service disruption, where it might get the system into a halt or unstable state, bound to the nature of the vulnerability and its exploit. Tests should be conducted before deploying any exploit. However, when this cannot be ensured a secondary propagation mechanism must be used.

4.6 Randomly Scanning Worm

We designed a simple random scanning worm to compare it with our mechanism in terms of bandwidth. Most worms released to counter malicious ones used random scanning (i.e. Welchia, Cheese). The comparison would help show if employing topology information would improve bandwidth utilization. Indeed, excessive bandwidth usage is observed frequently in worm outbreaks. The random scanning worm mimics the Slammer worm that uses a random scanning technique where it targets a randomly chosen IP, by sending one UDP packet to its victims [63] but with slight difference as the random scanning worm operates in corporate networks specifically at layer two of the OSI model.

When the worm is initiated, it propagates to all LANs in the corporate network according to a hard coded destination host at each LAN. Then, the randomly scanning worm will utilize the information in the host to indicate the address range it can randomly choose from. This is done by reading the IP address and the subnet mask, of the host. We have assumed for the simulation a class A network with subnet mask 255.255.192.0. The address space thereafter consists of 16384 IP addresses; ignoring the network address and the broadcast address yields 16382 possible target hosts.

The worm uses one packet of size 900 bytes to infect, and we assume all hosts are vulnerable. However, there are switches and routers that the worm might scan, with no infection as the vulnerability exists only in host nodes.

4.7 Simulation Results

Since each network has different topology and parameter choices, it is somewhat difficult to find a closed-form complexity estimate for the vulnerability discovery mechanism; we have therefore chosen to conduct extensive network simulations of our mechanism. For the results to be more realistic following the model and assumptions outlined in section 4.2, hierarchical networks were used, reflecting typical network topologies in large-scale enterprise networks.

All simulations have been conducted using the Network Simulator 2 (NS-2), a discrete event simulator mainly used for research activities [42]. The simulations in total were 765 running within hierarchical networks. Since the capacity of NS-2 is limited, we weren't able to conduct simulations on topologies over 8000 network nodes. Therefore, we have run our simulations against 100 to 8000 nodes with around 500 nodes difference between each topology.

For more accurate results each topology has been simulated 45 times where the average has been calculated to account for random effects such as link and node operation failure in addition to time coverage. In addition to Seawave's simulation, we have performed simulations of a random scanning worm following the model and assumptions outlined in section 4.6. However, due to the extensive amount of communications and bandwidth consumption of such random propagation strategy, the time of simulation was beyond our capacity, and therefore we settled down with one simulation of network topologies ranging from 100 to 3000 nodes.

In each simulation we have gathered the following:

- The number of nodes that were missed (i.e. not probed) by Seawave, due to a link failure of probability 0.01
- The number of packets generated by both the defensive worm and the randomly scanning worm to cover the network. The packets, however, that are generated between a switch and a host are exempted since they have no significant overhead on the bandwidth of the network. The packet size is 900 bytes.
- The time it took Seawave to cover the corporate network. Time is measured in simulation seconds and when there is a link failure, the mechanism will try to resend a packet after 2 simulation seconds.
- The number of random worm scans of non-existent IP addresses.

For the sake of simplicity, we have assumed in the simulation that the CAM table and STP information stored in the switch can be requested by sending one packet to the switch.

4.8 Discussion

Seawave in its first stage will be installed in the host dedicated to be the starting node. Since the directly connected switch is transparent, the Seawave agent does not actually see it. However, the agent can impersonate a switch and therefore can learn the designated switch MAC address from STP traffic. Then following the propagation algorithm outlined in section 4.4; the agent spreads around the network. Since switches do not necessarily have Reverse Address Resolution Protocol (RARP) [31] servers running on them (that maps the MAC address to its IP address), getting the IP address is not straightforward.

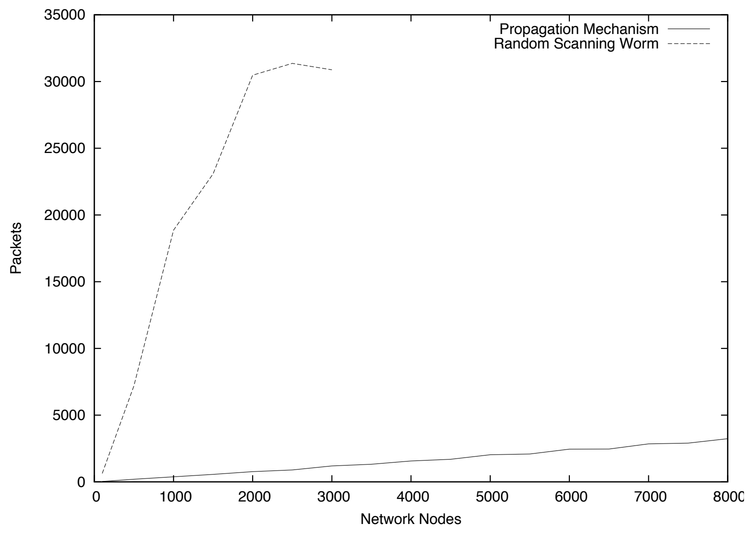


Figure 4.3: Packets generated by Seawave and the random scanning worm to cover the total network

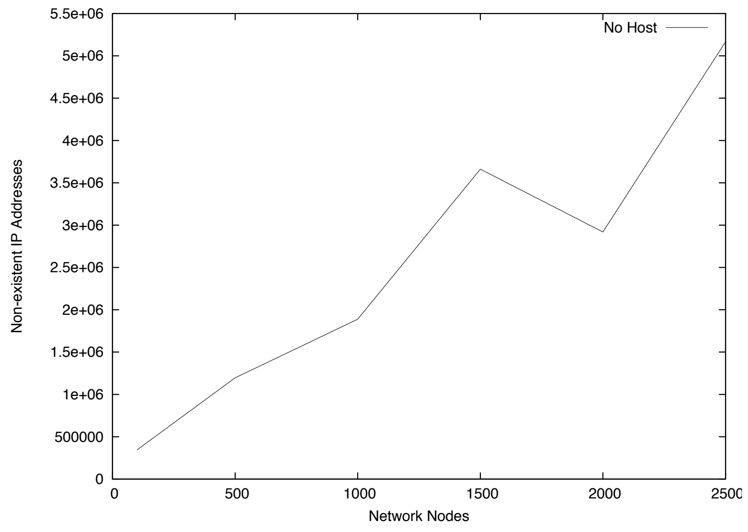


Figure 4.4: The random scanning worm number of scan attempts of non-existent IP addresses

Breitbart *et al.* [15] use SNMP MIB object *ipNetToMediaTable* in a single subnet switch domain to get the MAC after providing an IP address calculated based on network mask and IP address format of the router. Furthermore, Stott [95] listed other approaches which includes scanning all STP tables using every IP address in a given range and the Bridge ID

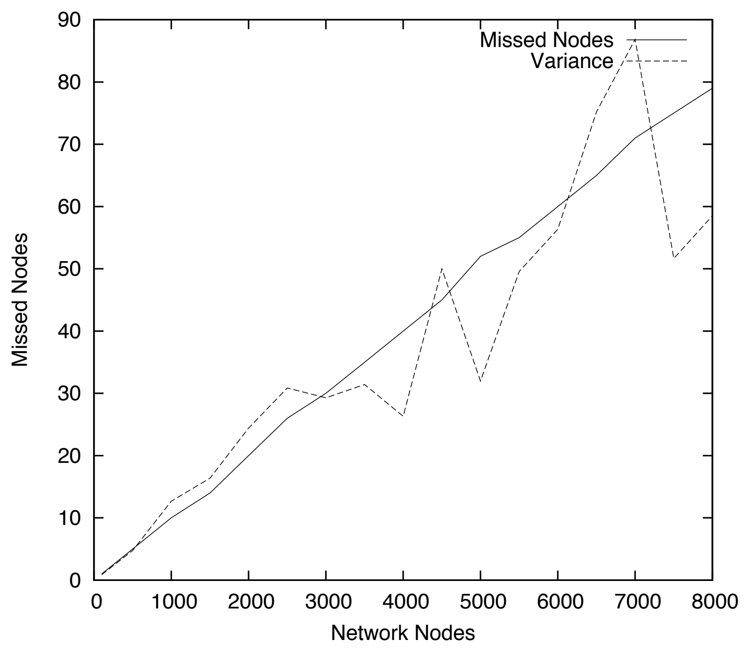


Figure 4.5: Missed nodes under link failure probability of 0.01

extracted from SNMP MIB object *dot1dBaseBridgeAddress* that matches the MAC would be the switch IP address. Another approach would be to scan switches to read the CAM tables searching for entries that matches the Bridge ID and whose SNMP MIB object dot1dTpFdbStatus is set to *self*, which means the address is assigned to the switch [95].

Since Seawave operates at layer two there is no necessity to use IP addresses, but this requirement arises when communicating through SNMP, as it operates at the application layer in the OSI model. In fact, the network interface card (NIC) would accept a packet with a broadcast IP address set as a destination. Therefore, assuming the switch does not have a firewall, one way to communicate is by using the switch's actual MAC address and the broadcast IP address of the subnet as a destination to allow the packet to reach its way to the application layer (SNMP server).

Seawave eases the impact on primary network links by depending largely on host to switch links to detect close targets which also provide short transmission distance for more efficient probing. According to simulation results

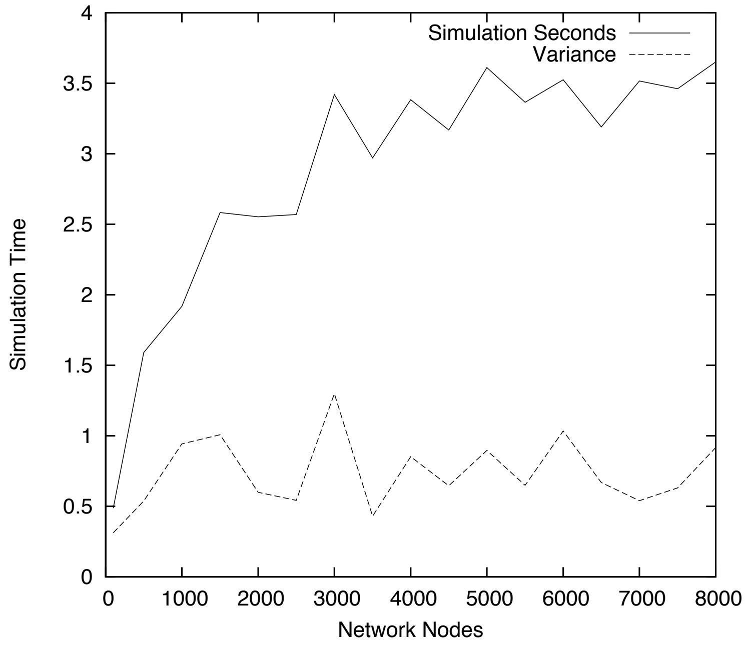


Figure 4.6: Simulation time required to cover entire network. After replicating failure, another attempt is triggered after 2 seconds.

Seawave generated 1564 packets to cover a network of 4000 elements, 2454 packets to cover 6000 elements, and 3232 to cover 8000 network elements. And in comparison to a randomly scanning worm, the defensive worm has generated 560 packets in a network of 1500 nodes, while the random scanning worm generated 23074 packets of the size 900 bytes, indicating the ability of Seawave to cover the network using 2.42% of the bandwidth used by random scanning. Moreover, the random worm generated 31358 packets to cover a network of 2500 nodes while our mechanism required 896 packets, about 2.8% of the random worm generated bandwidth. Further results are shown in Fig. 4.3.

The random scanning worm has generated too many scans to non-existent IP addresses. Although the worm operates within the LAN, non-existent IP addresses scans will cause the host to emit an ARP probe [80] for each new IP address scanning attempt. For a network topology of the size of 1500 it resulted in 3660723 non-existent IP address scan attempts. Furthermore, 5165367 nonexistent host scan attempts (or missed connections) were recorded for a network of the size 2500. Results are shown in Fig. 4.4.

Since the defensive worm depends on the up-to-date CAM and STP information stored in switches, missed connections are not common (i.e. probing a non-existent device) and the total bandwidth is used as effectively as possible without wasting communications. Seawave in its current form is probabilistic in that it does not ensure that all nodes on the switch have been probed for vulnerability (i.e. no failure recovery). Therefore, different missed nodes were reported under a link failure probability of 0.01. For instance, a network with 2000 hosts, 20 of them were missed and 52 nodes were unreached in a 5000 nodes network. The larger the network, the higher the rate of missed host nodes – more results can be seen in Fig. 4.5.

The time it takes the mechanism to cover the network is measured in simulation seconds. In a network of 1500 elements it took Seawave 2.6 simulation seconds to spread around the spanning tree and 3.6 simulation seconds to cover a 5000 nodes network. Of course the time is affected by link failures, the mechanism will wait 2 simulation seconds before trying to self-replicate after a link failure. Fig. 4.6 shows more results.

4.9 Summary

In this chapter we have proposed a controlled self-replicating, self-propagating, and self-contained vulnerability mitigation mechanism (Seawave) as one approach to probe for network vulnerabilities. The mechanism is topology-aware, which limits its bandwidth consumption and reduces the risk of uncontrolled propagation. Our defensive worm makes use of two types of information stored in switches: CAM table and Spanning Tree protocol. This topology information, enables Seawave to propagate around the network in a constructive manner, utilizing bandwidth efficiently to avoid any unnecessary communication.

Seawave adds some constraints to the self-replicating process by limiting replication to one node only directly connected to the switch and enabling that node to act as a vulnerability discovery entity to the remaining nodes directly connected to the same switch. The node would subsequently act as a *neighborhood watch* node, where the neighborhood consist of all nodes connected to the switch. In the chapter we presented the amount of bandwidth Seawave would generate in different hierarchical networks. The results also showed the amount of time – in simulation seconds – it took Seawave to spread around the network by following the spanning tree and under the link failure probability of $p = 0.01$.

The chapter presented the number of nodes that might be missed in the the process of vulnerability discovery. Also, bandwidth wise, we have compared our proposed vulnerability mitigation mechanism with a malicious worm that spread using random scanning. Ongoing and future work is focused on increasing both the robustness and performance of the algorithm in the face of intermittent bridging (e.g. found in wireless links) and defending the propagation mechanism against malicious adversaries.

Chapter 5

A Vulnerability Mitigation Worm – Seawave II

5.1 Overview

In this chapter we further enhance and improve Seawave I by adding edge node failure recovery, network backbone traversal, and intermittent node detection and recovery. These new features allow the defensive worm to ensure sound coverage by assuring each node has been probed for vulnerabilities and it also allows the mechanism to independently traverse the enterprise network without human intervention – Seawave I required human intervention – and it adds the capability to detect powered down, disconnected, or portable devices as soon as they connect to the enterprise network. Risks and threats to the mechanism are also addressed in this chapter.

5.2 Network Topology Model

The network model we have used for Seawave II follows the one described in Chapter 4, Section 4.2; however, to observe how the vulnerability mitigation mechanism deals with topology updates and intermittently active hosts, each node was set to run under two modes: *online* and *offline*. When the node is in the offline mode it can not be seen by the mechanism and is considered disconnected until it becomes online. Fig. 5.1 shows an example of a small 100-node network, with a switch and hosts in the offline mode. In the simulation we have assumed the enterprise network supports

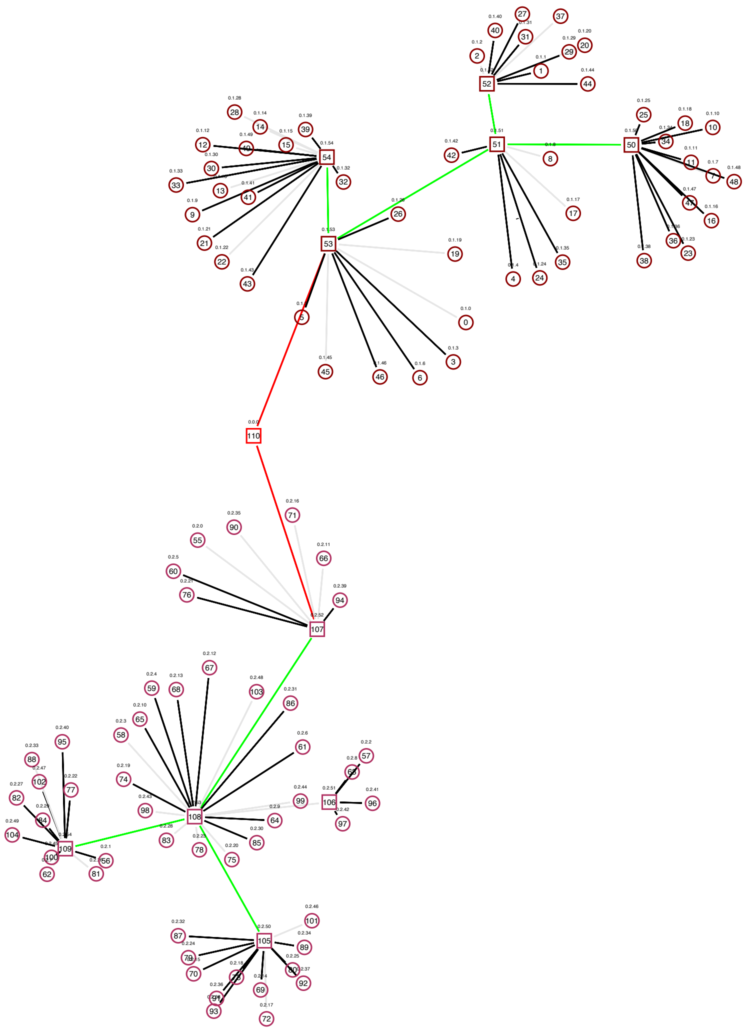


Figure 5.1: A 100 node hierarchical network that implements STP with scattered disconnected, i.e. offline, nodes (gray links), including a switch.

the following:

- Simple Network Management Protocol (SNMP).
- Spanning Tree Protocol STP and is loop-free.
- Open Shortest Path First (OSPF) [67, 68] as a routing protocol.
- CAM and port status is maintained by switches.

We have generated 17 hierarchical networks consisting of offline nodes and switches; nodes are linked via a duplex-link where packets can flow in both directions, and for the results reported in section 5.7, the number of nodes was chosen between 100 and 8000, with link bandwidth set to 100 Mbps and assuming a discovery packet size of 900 bytes. If we consider a switch domain as a switch with its directly connected hosts, then there exist three failures that Seawave II might encounter, including:

- Self-replicating failure to the next switch domain.
- Failure when probing an edge node.
- Self-replicating failure to the next LAN.

The cause of failures, however, is either due to a link or a node operation failure. In both cases the agent will try to recover the failure. More simulation attributes are mentioned in section 5.7.

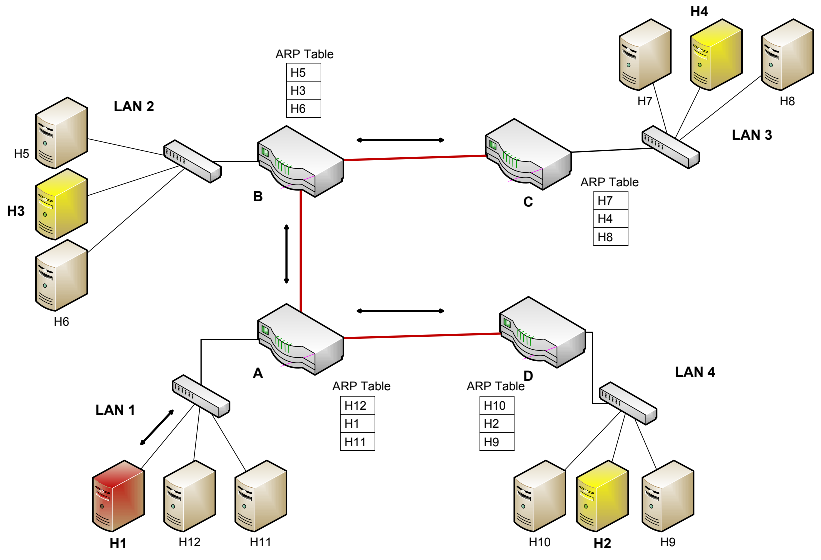


Figure 5.2: Traverse Backbone Algorithm

5.3 Seawave II

The vulnerability mitigation mechanism proposed in Chapter 4 (Seawave I) falls short in performing edge node failure recovery and topology change detection, which is detrimental when confronted with volatile edge networks. It also does not provide an algorithm for traversing the network backbone, which requires assigning a host at each LAN as a starting point. Therefore, in this section we provide enhancements to address these limitations.

In order to provide edge node failure recovery, Seawave will wait for a response from edge nodes to ensure successful probes; upon failure to respond, the agent will retry after 0.5 seconds. For the vulnerability mitigation agent to be able to propagate from one LAN to another, it needs to map the topology of the backbone. Since the OSPF routing protocol is implemented, the agent upon detection of a router during its propagation, sends to the router an SNMP MIB request to fetch the OSPF Link State Database (LSD), which gives a complete description of the backbone, including: Routers, network segments, and how they are interconnected [69]. This database is built by the collection of Link State Advertisements (LSA) sent by each router in the backbone to describe its local routing information. The agent also reads the ARP cache of the LAN interface of each router to detect a host IP address in order to self-replicate to. For example, Fig. 5.2 shows a sample of a network of four LANs connected by a backbone that consists of four routers. When the agent reaches $H1$ and detects a router device (e.g. by checking the SNMP MIB value of *ipForwarding* if set to one then the device is a router [15]), it will send an SNMP MIB request *ospfLsdbTable* to fetch the LSD [69] and the ARP cache stored at router A using SNMP MIB *ipNetToMediaTable*; upon receiving router's A response the agent at $H1$ extracts the backbone network map, which in this example consists of routers A,B,C, and D. The agent thereafter sends an SNMP MIB request to routers B, C, and D to fetch the stored ARP cache of network interfaces connected to the LAN. In parallel, the agent picks a host in the ARP cache table of router A to detect a valid IP address that belongs to a LAN directly connected to the router for the agent to propagate (No IP will be picked in the case of the example, since there is only one LAN connected to router A). The agent subsequently attempts to self-replicate to the selected IP address and waits to receive replies from routers B, C, and D containing ARP cache tables. Upon receiving the tables, the agent picks one IP address for each LAN and self-replicates to it; to ensure comprehensive coverage of all LANs in the enterprise network.

We denote the list of LANs connected to the first router encountered by the mechanism as $L(ARP_{R_0})$ where ARP_{R_0} is the ARP cache stored at router R_0. And we denote LSD stored in the first router as LSD_{R_0}, where the number of routers according to the database is $LSD_{R_0}\{N\}$. We denote the function of fetching data through SNMP as *SNMP* and agent self-replicating as *SR*. The algorithm for traversing the backbone, then follows:

1. Fetch Link State Database (LSD) and ARP cache from the first encountered router, using SNMP: $SNMP\{LSD_{R_0} \cup ARP_{R_0}\}$

2. Pick an IP address from each LAN interface (else the source LAN interface) in the ARP cache of the first encountered router and self-replicate to that IP address: $SR(L(ARP_{R_0}) - \{L_0\})$

3. Fetch ARP cache from all other routers in the backbone (according to LSD), using SNMP: $SNMP\left\{ARP_{R_1} \cup ARP_{R_2} \cup \ldots ARP_{R_{LSD_{R_0}\{N\}}}\right\}$

4. Pick an IP address from each LAN interface in the ARP caches of backbone routers and self-replicate to these IP addresses: $SR(L(ARP_{R_1}) \cup L(ARP_{R_2}) \cdots \cup L(ARP_{R_{LSD_{R_0}\{N\}}}))$

The algorithm assumes that the autonomous system consists of a single area only, where a single read of the LSD is enough to determine the map of the backbone. Dividing the backbone into more than one area is used in very large networks to reduce the size of routing tables, but can be handled in a straightforward manner.

5.4 Intermittently Active Hosts and Topology Changes

In dynamic networks, it is difficult to achieve efficient coverage of a network with transient nodes without excessive scanning frequency. However, by utilizing topology information it is possible for Seawave to determine topology changes and intermittently active hosts for vulnerability discovery. The Seawave agent takes advantage of the CAM table stored in the switch and stores it during the first scan round. After 0.5 simulation seconds, the switch is probed for the CAM table and checked for changes; new nodes or switches are thereafter detected by the agent and probed.

When a new switch is listed in the CAM table the agent has to verify that it is actually a directly connected switch, by using *Lemma 4.1* and *Lemma 4.2*. Intermittently active hosts and topology changes detection can, therefore, be achieved by the following algorithm:

1. Fetch CAM table from the directly connected switch.
2. Compare the CAM table with the locally stored previous CAM table.
3. Probe newly detected edge nodes.
4. Probe newly detected switches devices according to *Lemma 4.1* and *Lemma 4.2*
5. Go to step 1 each time t

Note that to speed up the simulation, we have chosen 0.5 simulation seconds for the agent to check the CAM table. However, the interval may be up to 5 minutes (the default expiration time for CAM tables stored in switches) [86]. But what about systems missing from the CAM table due to inactivity or table age expiration? Those systems will be treated as offline nodes and will be detected when they become active as per the algorithm.

The agent can also check for topology changes by listening to BPDUs emitted from the root switch. When a topology change occurs, i.e. when a switch port goes into forwarding status or from forwarding (or learning) into blocking, the switch will emit a *Topology Change Notification (TCN) BPDU* to the next switch (towards the root bridge) until the root switch receives the *TCN BPDU* and emits a configuration BPDU with the topology change bit set. The BPDU emitted by the root is sent to all switches and thereafter the agent can listen to it, where it can trigger a topology change detection. However, in case the agent might miss a topology change BPDU we have set the agent to check for network topology changes each 0.5 simulation second.

5.5 Design Components of Seawave

Nazario *et al.* defined different components that constitute a worm system [72]. We use their components to describe the design of our mechanism with slight modifications. Seawave consists of three components that allow it to cover the network as follows:

- *Reconnaissance.* This component is responsible for discovering host nodes that are vulnerable. Seawave II reads CAM tables stored in switches to detect edge node hosts to probe for vulnerabilities. The mechanism also reads STP in addition to the CAM table to determine next switch to propagate to.

- *Probe Component.* This component describes the method Seawave's uses to detect a vulnerability at a target node. For the sake of simplicity, our simulation assumed all hosts to be susceptible and it requires only one packet of the size 900 bytes to exploit a vulnerable node.

- *Communication.* This component describes the communication between agents. Seawave II enables communication between agents by sending an acknowledgment packet to the sender agent to only ensure that the self-replicating task has been accomplished successfully so as to disable any blocking attempts.

These three components summarize the design of the vulnerability mitigation mechanism; further extensions are discussed briefly in section 5.9.

5.6 Risks and Threats

Since the propagation path depends on topology information, any malicious interactions with STP, OSPF, ARP, SNMP, or CAM tables might have negative impact on Seawave's behavior. For example, the lack of authentication of the STP protocol, makes it possible for a malicious user to manipulate the topology and compromise the integrity of BPDUs leading to undesirable actions such as changing switch port status or electing a compromised switch as root. Yet, although these vulnerabilities do not relate to the agent directly, any countermeasure to prevent such abuse would participate in the protection of the defensive worm itself. Some protection measures do exist for STP, such as disabling user ports upon detection of STP traffic and disabling ports that emit false BPDUs that elect false roots.

A malicious user flooding a CAM table with fake MAC addresses, will cause the switch to act as a hub, with no edge information for the agent to utilize. One possible mitigation to such misuse would be shutting down a port if more than one MAC were detected, performance issues however, should be considered when applying such mitigation.

For the agent to traverse the backbone it has to read the ARP cache stored in the router. ARP cache poisoning may redirect the agent to an IP address – out of Seawave's scope – inserted by malicious users, however, the agent can be designed to ignore any IP address that do not adhere to certain attributes put by the enterprise security team. ARP cache poisoning can be achieved by sending malformed gratuitous ARPs to the target machine, however, private VLANs can – to some extent – eliminate such abuse, as they do not allow nodes on different ports to communicate at layer two but still allow them to share the same network space.

Since the agent uses SNMP in its communication with network devices (e.g. switches and routers) a malicious user can intercept this line of communication and alter it according to its attack preference. When the agent, for example, probes a neighboring switch to determine if it is directly connected to the current switch, the attacker can respond with an SNMP message that indicates the switch to be a non-neighbor switch. This will cause the agent to ignore the switch, and might cause the agent itself to stop any further propagation. Another scenario, would be an attacker altering the router response to fetch the OSPF Link State Database SNMP request, giving the attacker the freedom to define the backbone map according to his intrusion preference and using it as an input to the mechanism.

Even if we assumed the agents to not be vulnerable to STP, ARP, or SNMP attacks; a switch domain can be isolated from the vulnerability discovery process by turning the agent off physically (weather the intention malicious or due to human error). Also, the agent can be controlled remotely if it was running under a vulnerable operating system where an attacker can exploit or if the intruder was able to compromise a privileged user account on the agent host. To reduce these threats, different protective measures and performance aspects of Seawave are addressed in Chapter 7 followed by a formal threat analysis model of Seawave in Chapter 8.

5.7 Simulation Results

Based on the simulation aspects and the random scanning model described in Chapter 4 section 4.7 and 4.6; in addition to the assumptions mentioned in section 5.2, we have run our simulations and gathered the following results:

- Number of link failures under $p = 0.01$.

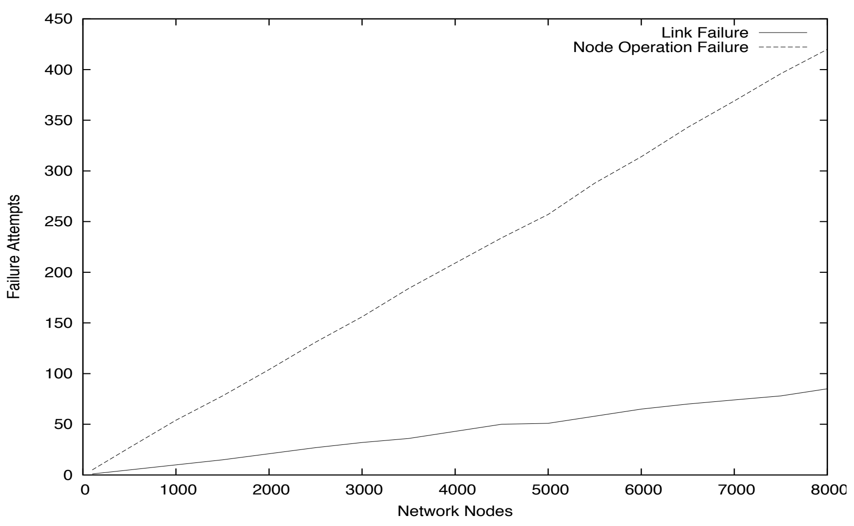


Figure 5.3: Failure due to link ($p = 0.01$) or node operation ($p = 0.05$).

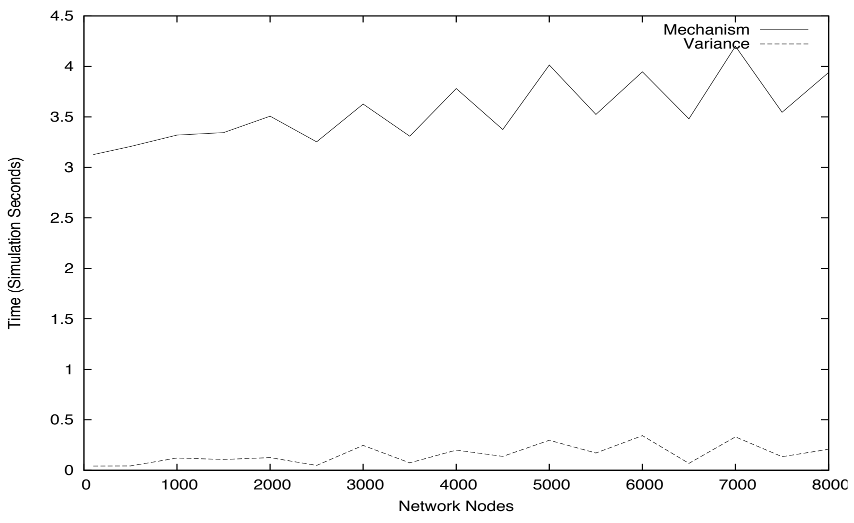


Figure 5.4: Time to cover the whole enterprise network

- Number of node operation failures under $p = 0.05$. Note that node operation failures are caused by the node itself (e.g. system is busy or in different state due to restarting).

- Number of packets generated by both Seawave II and the randomly scanning worm to cover the network. The packets, however, that are generated between a switch and a host are exempt due to them having

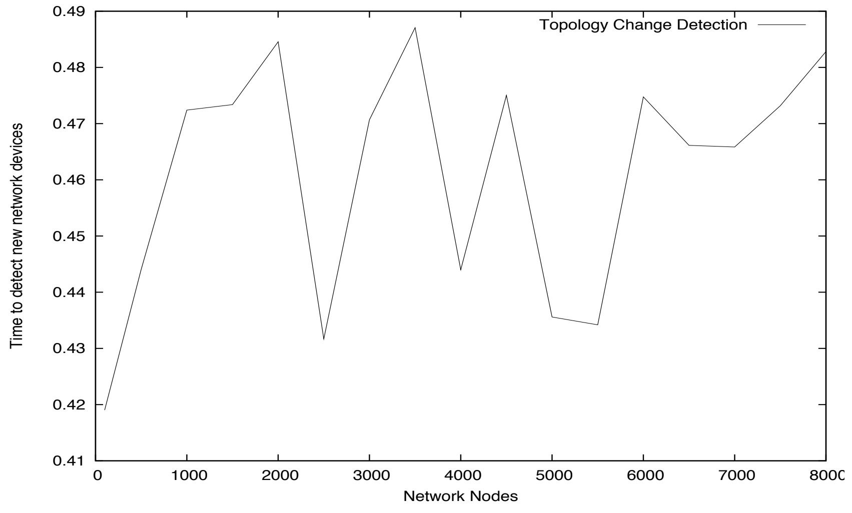


Figure 5.5: Time to detect all newly added network devices

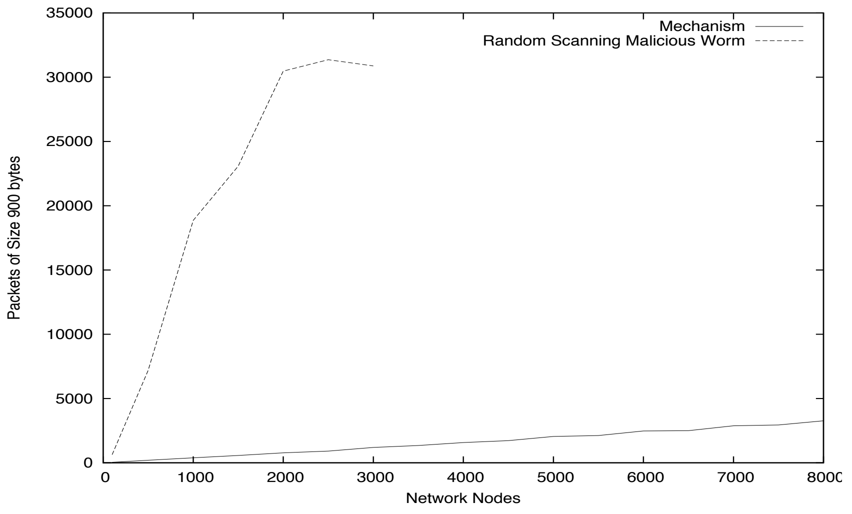


Figure 5.6: Number of packets generated by randomly scanning worm in comparison to our scanning mechanism

no significant impact on the bandwidth of the network.

- Time it takes the mechanism to cover the corporate network. When there is a link or a node operation failure, the mechanism will retry after 0.5 simulation seconds.

- Time it took the mechanism to detect all new network nodes that have just joined the network. In all simulations the network nodes were ran-

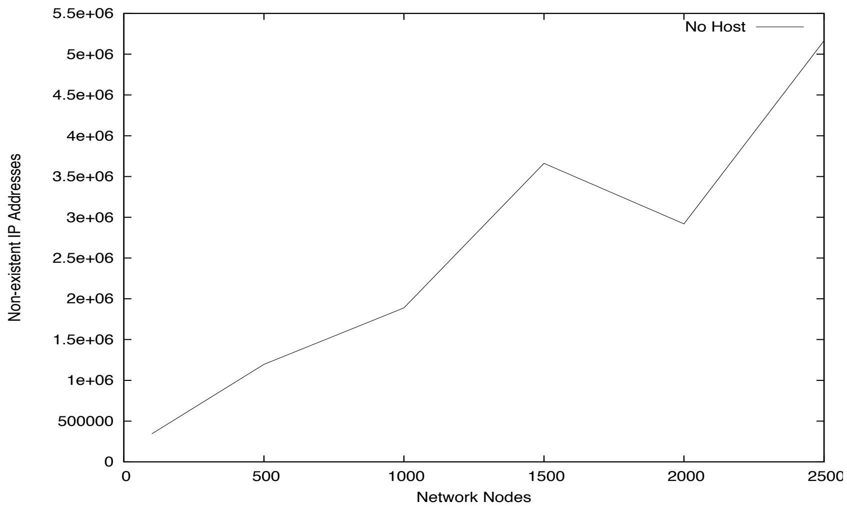


Figure 5.7: The random scanning worm number of scan attempts of non-existent IP addresses

domly added to the topology at different locations. All offline nodes became online at the 2.5 second of the simulation time.

- Number of random worm scans of non-existent IP addresses.

For the sake of simplicity, we have assumed in the simulation, that the CAM table and STP information stored in the switch can be requested by sending one packet. A single SNMP MIB request is also assumed to fetch OSPF information in addition to ARP cache table stored in the router.

5.8 Discussion

Seawave tries to combine both the distinctive exploring nature of self-replication programs and the constraints of the enterprise network, to provide a sound protection without disturbance to the network focal interests. It appears that by using network topology information it is possible to dramatically reduce the bandwidth utilized by self-replicating programs as in the case with Seawave II.

Seawave I (see Chapter 4) had some limitations in its capability to spread around the network and achieve sound coverage without human intervention (i.e. hard coding a single host IP at each LAN to bypass the backbone). The improvements thereafter are required to address these limitations to produce a more robust vulnerability discovery mechanism. Three limitations of the previous approach have been addressed by Seawave II. First, it adds edge node failure recovery ensuring that all network nodes are not missed; this is important since one vulnerable node can cause a threat to the whole enterprise network. Second, it provides an algorithm to detect newly added network devices, which ensures that a vulnerable node joining the enterprise network get probed as soon as possible to minimize the time of vulnerability exposure. Third, it provides an algorithm to bypass the network backbone utilizing protocols such as OSPF and ARP; which gives the mechanism the capability to spread and cover the vulnerable network as fast as possible to achieve absolute protection.

As in any mechanism Seawave is not fully immune to threats and risks. The mechanism depends on topology information to propagate, which leaves it bound to the protection measures applied to this information. Vulnerable STP allows a malicious user to control and define the path the mechanism would take; this could allow the attacker to hide nodes from the mechanism, or even cause a denial of service attack by adding network loops to the topology. Notice that this is not a direct attack on Seawave itself but on the feedback the mechanism uses to function. The same applies to the CAM table, what if a malicious user alters it to reflect no switches available. This would lead the mechanism to find no further way to propagate and stop. More threats exist in the backbone as well.

Seawave depends on OSPF protocol to pass to other LANs, what if an attacker was able to forge the LSD to contain non-existent IP addresses? This will stop the vulnerability discovery process from reaching further LANs. Or even worse, the compromise of the LSD would allow the attacker to define routers IP addresses that is not necessary part of of the predefined scope of the mechanism. Likewise with ARP cache stored at routers, a poisoned ARP cache can give the attacker the ability to direct the worm to self-replicate to whatever IP is given as long the target is vulnerable. This would take the vulnerability mitigation mechanism out of its predefined scope and would add a malicious intention to its operational procedure.

Detecting new switches and nodes is also threatened by malicious activity. One scenario would be adding a rogue switch where Seawave would not be able to deal with. When Seawave probes the rogue switch for CAM or STP information, the switch would provide misleading response, such as fake designate bridge, allowing the switch to remain with its directly connected malicious nodes away from the mechanism's coverage. Seawave assumes the topology information to be authentic and has not been altered. However, if that is not the case, it becomes vulnerable to the same vulnerabilities the network topology has. This is expected as the mechanism in its topology dependent nature becomes part of the network, like the protocols operating within the network, not like an independent external security system.

The simulation results cover different aspects of performance. Node operation and link failures of $p = 0.05$ and $p = 0.01$, respectively, have recorded different results. For example in a network of size 2500 nodes, link failures were 27 and node operation failures were 131. Further, 85 link failures were recorded at a network of the size of 8000 nodes and 420 node operation failures at the same network. Fig. 5.3 shows more results. The time it took the agent to cover a network of 2500 elements was 3.25 simulation seconds and 4.0s to cover a 5000 network topology. Of course the time is affected by link and node operation failures, Seawave will wait 0.5s before retrying after a link or node operation failure. Note that the time reflects the coverage of the whole network including newly added network devices, that join the network at simulation second 2.5. Fig. 5.4 shows more results.

For the topology change detection, which records the time it takes the propagation mechanism to detect all newly added network devices, it shows close results. That is because each topology detects almost the same number of network devices. In a network of the size 3000 it required 0.47s to detect 30 hosts and a switch (with different hosts connected to it) and 0.48s for a network of the size 8000. More results can be seen at Fig. 5.5. In comparison with a randomly scanning worm, our mechanism has generated 570 packets in a network of the size 1500, while the random scanning worm generated 23074 packets. Moreover, the random worm generated 31358 packets to cover a network of the size 2500 while our scanning algorithm required 912 packets only. Further results are shown in Fig. 5.6.

The random scanning worm has generated too many scans to non-existent IP addresses. Although the worm operates within the LAN, non-existent IP scans will cause the host to emit an ARP probe for each new IP address scanning attempt. For a network topology of the size 1500 it resulted in 3660723 non-existent IP scan attempts and 5165367 for a network of size 2500. Results are shown in Fig. 5.7. Seawave, however, avoids scanning non-existent IP addresses since it detect targets based on CAM information.

5.9 Summary

We have previously proposed a defensive worm (Seawave I) to help prevent malicious attacks in Chapter 4, and in this chapter we further enhance its performance. Improvements include: edge node failure recovery to ensure sound and efficient coverage of all vulnerable edge nodes and an algorithm to enable the mechanism to traverse the network backbone using OSPF protocol and ARP stored in routers – which provides independently driven self-propagation to all network LANs without human intervention as compared to the previous design.

We also proposed an algorithm to enable the mechanism to detect newly added network devices as soon as possible to eliminate vulnerability exposure. Since Seawave is topology dependent, different risks and threats to topology information have been highlighted, especially protocols utilized in the discovery process, such as: STP, OSPF, ARP, SNMP, and CAM.

We have validated Seawave through simulations; the results shown here assumed relatively volatile link failures of probability $p = 0.01$ and node operation failure of $p = 0.05$. The results also showed the time it took Seawave to cover the network, in addition to the time required to detect newly vulnerable nodes that have just joined the corporate network. Simulation results of a randomly scanning worm that to some extent mimic the propagation behavior of the Slammer worm have also been gathered to be compared with the vulnerability mitigation mechanism in terms of bandwidth utilization. Ongoing and future work is focused on increasing both the robustness and performance of the algorithms in defending the propagation mechanism against malicious adversaries.

Chapter 6

An LLDP Based Vulnerability Mitigation Worm

6.1 Overview

In this chapter we propose a simple defensive worm that traverses the enterprise network with only knowledge of the immediate network neighborhood as can be obtained from passive observation of the LLDP protocol. This minimizes bandwidth consumption in conjunction with persistent agents deployed by the traversal to capture transient or intermittently active nodes. We also analyze the efficiency of our propagation mechanism under different topologies, while considering – in our simulations – link, node, and discovery attempts failures. We also compare the mechanism with blind vulnerability discovery and Seawave I[1].

6.2 Introduction

Seawave depends mainly on the spanning tree protocol, but for networks that do not support STP, a defensive worm can look for alternative protocols to get topology information. In this chapter we propose a mechanism for rapid automated vulnerability discovery and mitigation dissemination using local knowledge of network topology to both contain propagation and – more importantly – to reduce communication complexity while retaining

[1]Although, in this chapter we propose another vulnerability mitigation worm, for the sake of this thesis, we concentrate our work on Seawave; and leave further improvements of this defensive worm for future work.

propagation speed and robustness. Our mechanism utilize the Link Layer Discovery Protocol to learn about neighboring nodes and self-propagate gradually until total coverage of the enterprise network.

6.3 Network Topology Model

The topologies considered in this chapter assume a hierarchical structure as typically found in structured (enterprise) networks; we formally model these topologies using the Transit-Stub model based on work by Zegura [113], obtaining hierarchical graphs by composing interconnected transit and stub domains. The network is developed by initially constructing a connected random graph then each node is replaced by another randomly connected graph representing the backbone of the network. Each node in the backbone is then replaced by a randomly connected graph to represent a LAN connected to a backbone node. Additional edges are then added within LANs and backbone with edge probability 0.5 within Local Area Networks (LANs) and edge probability 0.8 between each pair of backbone nodes [113]. We assume each node in the topology (including routers) supports LLDP. Five different topologies were generated. For topologies that support STP (where Seawave is due to run), we follow the model described in Chapter 4 Section 4.2.

For more accurate results different numbers and layouts of hierarchical networks have been designed for simulations. These hierarchical topologies consist of a backbone with different number of LANs connected to it. The Drop Tail queue management algorithm has been used as a queuing algorithm. Nodes are connected via a duplex-link where packets can flow in both directions. The number of nodes vary between 72 to 260 with a link failure probability of $p = 0.01$ and node operation failure of $p = 0.05$. In each simulation, the initial node triggers the vulnerability discovery mechanism by sending a packet of 900 bytes within 0.2 seconds of starting the simulation.

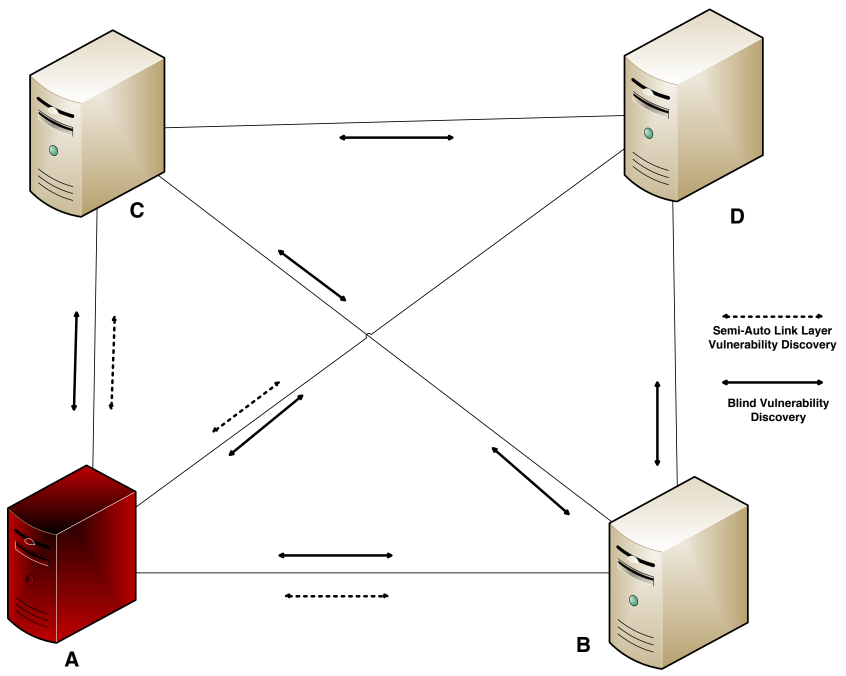

C
D
A
B
Semi-Auto Link Layer Vulnerability Discovery
Blind Vulnerability Discovery

Figure 6.1: Zero-Topology knowledge vulnerability discovery (blind)

6.4 LLDP Based Vulnerability Mitigation Worm

In dynamic networks or networks where the topology is unknown or always changing it is difficult to achieve efficient coverage . However, with some limited knowledge of network topology, a vulnerability mitigation mechanism can find its way through the network in a robust manner. If we assume that each node in a hierarchical network knows its adjacent nodes (neighbors), it would be possible for many nodes to avoid probing an already probed system, which – compared to blind vulnerability discovery – reduces the traffic load on the network and minimizes processing time.

Our proposed algorithm is a vulnerability mitigation worm, which utilizes information from the data link layer (layer 2 of the OSI model) to reconstruct topology information found through the *Link Layer Discovery Protocol* to detect neighboring nodes and propagate gradually until total coverage of

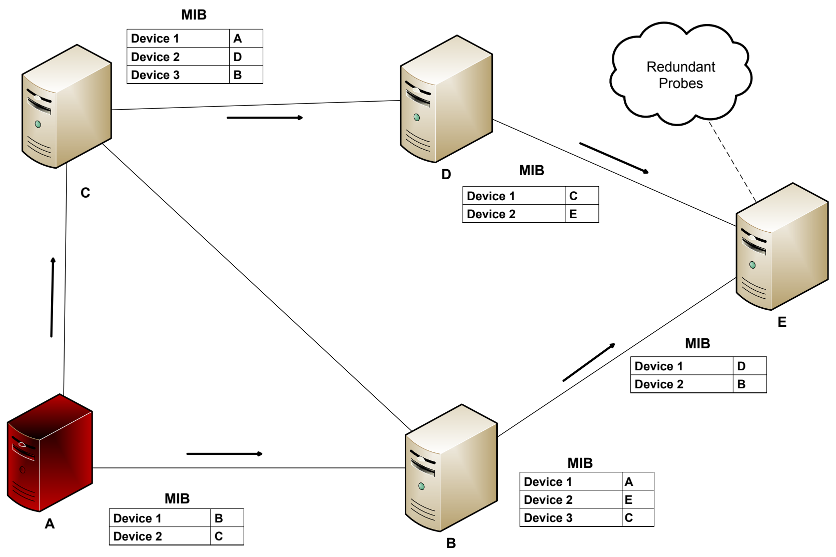


Figure 6.2: LLDP Based Vulnerability Mitigation Worm

the enterprise network.

In a fully connected four vertexes graph, such as in Fig. 6.1, each node will probe its neighbors. For example, if A started probing, it will probe node B, C, and D. Node B will probe node C and D. Node C will thereafter probe B and D. Node D will probe its adjacent neighbors C and B. Nodes afterwards will respond to other probes in a similar way, which results in six redundant probes and four transaction (assuming equal processing time for each probing and forwarding step).

However, when the immediate neighbors are known, there will be no redundant probes and only a single transaction step is required. We note that the neighborhood information is already provided by the *IEEE Link Layer Discovery Protocol* (LLDP) and can be used to discover adjacent network nodes without incurring additional cost. LLDP is a *media independent protocol intended to be run on all IEEE 802 LAN stations and to allow an LLDP agent to learn the connectivity and management information from adjacent stations.* [40]. The mechanism traversing the network takes advantage of the flexibility of self-replicating and self-propagating programs (agents) to distribute the vulnerability discovery process around the network without regard to network topology. The use of agents enables the mechanism to capture intermittently active nodes and probe hosts from closer distances, easing the impact on busy links; for more features see Chapter 2 Section 2.1.

When a node gets probed and an agent is installed it checks to see if there exist any common neighbors with the probing node, and probes all adjacent nodes except common neighbors. For example, node A probes the adjacent node B, then the agent at node B compares its neighbors list $N(B)$ with A's neighbors list $N(A)$ and probes $\{N(B) \backslash N(A)\} - \{A\}$ to avoid duplicate probes if possible. However, since different nodes can share the same neighbor it is possible for a node to receive redundant probes or to initiate unnecessary transactions. For example, in Fig. 6.2 when node A initiates agent propagation, there will be two propagation paths. The first path passes through node C, D, and E, while the other path pass through B and E. Node E, therefore, will receive two redundant probes as both paths leads to E. However, the existence of more than one path to probe a node has the positive side of the mechanism being able to reach the node even if a specific path has been blocked. Multiple paths helps in preventing malicious attempts directed towards stopping the mechanism from propagating, such as a rogue node that does not react normally to common network behavior. Furthermore, multiple paths allow the mechanism to avoid missing large parts of the network when a certain path is not responding positively to agent propagation (i.e. parts that can not be reached from path A can be reached by path B). More threats on the mechanism are set for future work.

6.5 LLDP Based Vulnerability Mitigation Defensive Worm Implementation

Based on LLDP, each node will store a local data base known as Management Information Base (MIB) which lists neighbors connected to the node.

The database can be accessed by requesting LLDP MIB objects using SNMP. The nodes advertise information about themselves to their neighbors and collect the information they receive about their adjacent nodes. When a node needs to learn the list of neighbors of a sender, it accesses the sender's MIB through SNMP. When node A probes node B, node B communicates with node A through SNMP to obtain the list of neighbors, if this information is not already retained locally. Regardless of the preceding step, B can then iterate through the list of neighbors in sequence and probe all neighbors other than common neighbors. This is illustrated in Fig. 6.2 as node A triggers the mechanism where it starts by reading the station's neighbors, which are in this case nodes B and C. When the agent self-replicates to node B and C both stations will start reading their own neighbors and propagate. Node B will read A, C, and E and will then read the neighbors list of the source by requesting the LLDP MIB stored at node A if necessary, obtaining nodes B and C. B, thereafter, will ignore node A because it is the source node and will ignore node C because it is a common neighbor between A and B and will self-replicate to node E. A similar scenario will occur to node C, where it will self-replicate to node D (ignoring A and B) and then to node E. Node E, however, will receive two redundant probes that is due to node E being a common neighbor to two nodes whom are not adjacent (B and D).

Assuming a node that has received the agent from another node as $n_{receiver}$ and the node that sent the agent as n_{sender}. The algorithm can be summarized as follows:

1. Install agent at the starting host n_0 in a subnet.
2. Unless $n_{receiver} = n_0$, if $n_{receiver}$ is already probed then stop, else continue.
3. Agent reads $n_{receiver}$ LLDP Management Information Base objects to extract adjacent neighbors A.

4. Agent then reads n_{source} LLDP Management Information Base objects to extract adjacent neighbors B.

5. Remove $n_{receiver}$ and n_{sender} from the lists and compare them and self-replicate to non-common neighbors, that is $\{A \backslash B\} - \{n_{source}\}$.

6. Go to step 2

We assume that adjacent neighbors are nodes directly connected to the sender or receiver nodes. We consider the whole network as N. The agent propagates through adjacent nodes sets $\{A_0, A_1, A_2..\}$ where $N = \{A_0 \cup A_1 \cup A_2..\}$. When the intersection of any two sets (whom are not adjacent) is not empty then redundant probes would occur, that is $\{A_0 \cap A_1\} \neq \phi$

Agents play an important role in achieving sound coverage of the network. When a node is offline it would not appear in LLDP list of adjacent nodes, allowing the vulnerability discovery mechanism to fail to see the offline node. However, when each agent around the network reads neighbors list periodically, nodes that go online or offline would be noticed and suitable actions can take place. As soon as a node appears online, it announces itself through LLDP, where an agent will be able to detect it and thereafter probe it for any vulnerability – assuming the vulnerability mitigation techniques described at Chapter 4 Section 4.5. In addition, agents would help to address the problem in relation of network dynamics, the existence of links and disappearance of others. When there is a new segment connected to the network, as long as LLDP can be read, it can be detected by agents and thereafter get assessed for vulnerabilities, reducing the time of vulnerability exposure as much as possible.

To avoid any possible unauthorized alteration of vulnerability discovery packets, integrity protection measures can be applied. Both message integrity and authenticity are required to provide secure vulnerability probes; one way to provide this is by using Message Authentication Codes or MACs, a symmetric technique that depends on a secret key distributed among communication entities. It can be derived by block ciphers or cryptographic hash functions. When the sending and receiving nodes generate a valid MAC then they can build a secure line of communication. SSL/TLS and IPSec for example utilize such technique [37]. Each MAC is embedded into a packet, by running the key against the packet, the receiving node can verify message integrity and origin.

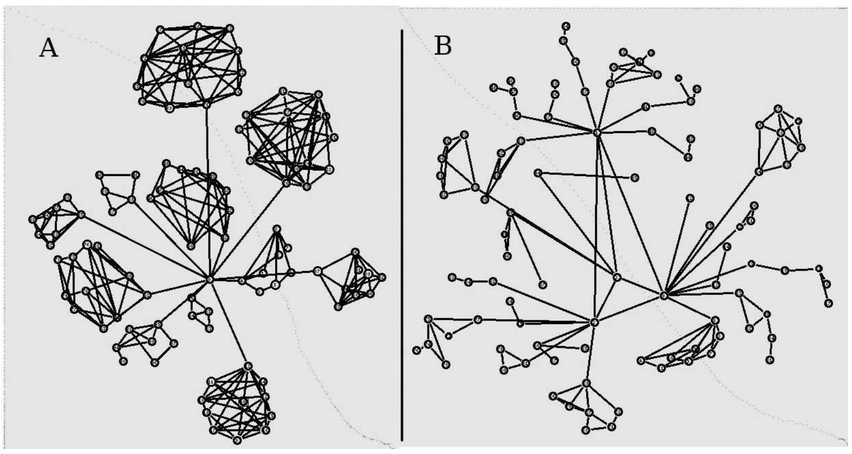


Figure 6.3: Two network topologies with the same number of nodes (100), but with different layouts.

6.6 Vulnerability Discovery Mechanism Design Components

Nazario *et al.* defined different components that constitute a worm system [72]. We use their components to describe the design of our LLDP based defensive worm with slight modification. The vulnerability discovery mechanism consist of three components that allow it to cover the network as follows:

- *Reconnaissance.* This component is responsible for discovering host nodes that are vulnerable. Our discovery and propagation mechanism depends on the information provided by the LLDP protocol to propagate. Hosts are discovered by looking up the neighboring nodes stored in the LLDP database.

- *Probe Component* This component describes the method the mechanism used to detect the vulnerability at a target node. For the sake of simplicity, our simulation assumed all hosts to be susceptible and it requires only one packet of the size 900 bytes to exploit another node and we assume vulnerability mitigation techniques as described in Chapter 4 4.5. However, in a heterogeneous network, more elaborate scanning will be required from agents.

- *Communication.* This component describes the communication between agents. The scanning mechanism does not allow communication between agents at this stage, else for detecting if the node has been probed more than once.

These three components summarize the design of the vulnerability mitigation mechanism; further extensions are to be added in future work.

6.7 Risks and Threats

The vulnerability mitigation mechanism assumes the topology information to be authentic. However, if that is not the case, it becomes vulnerable to the same vulnerabilities the network topology has. This is expected as the mechanism in its topology dependent nature becomes part of the network, like the protocols operating within the network, not like an independent external security system.

When a malicious entity compromises a network node, it will have the ability to alter the LLDP database stored in the system. Which gives the attacker the force to at least hide the compromised node from the mechanism's sight; that can be achieved by stopping the node from advertising its identity. However, even though the compromised node has hidden itself from vulnerability detection, its neighbors most probably have been detected by the mechanism, which leaves the vulnerable node to some extent isolated in the network. Exceptions exist if the neighbors of compromised node are only connected to it, which leaves the node and its neighbors vulnerable.

When an attacker compromises an agent itself, he can stop the propagation of the mechanism by deleting the LLDP database providing no further hosts to scan. However, other agents distributed throughout the network might be able to cover the vulnerability gap caused by such malicious activity. For the attacker to stop the whole mechanism from vulnerability detection he has to compromise each agent, which is a difficult task unless the malicious user was able to control the agent(s) in the very early stages of propagation.

6.8 Simulation Results

In order to measure the performance of both the baseline blind scanning and our proposed vulnerability discovery mechanism, computer network simulations were used. All simulations have been performed using Network Simulator 2 (NS-2) [42]. As described in section 6.3, hierarchical networks were generated to model larger enterprise networks. Two parameters were considered in simulations: The number of network nodes and different network architectures. In total, there were 300 hierarchical network simulations performed by NS-2. The simulations were grouped into 5 groups each group consisted of a different quantity of nodes that varied from 72 to 260. For blind and Seawave mechanisms, 5 different layouts of the same topology were generated. Note, blind and LLDP mechanisms were run under the same topologies, while Seawave was run under topologies that support STP. In each simulation we have gathered the following:

- The number of link failures under the probability of 0.01.

- The number of node operations failures under the probability of 0.05. Node operation failures are failures caused by the node it self (e.g. system is busy or in different state due to restarting).

- The number of redundant probes issued by each mechanism. Redundant probes are probes received by a node more than once.

- The actual number of missed nodes during the process of vulnerability discovery.

The simulations were run at each group and the average result was calculated for better accuracy. Toplogy nodes consisted of router nodes on the backbone and host machines spreads around the network as LANs.

6.8.1 Worm's Sensitivity to Network Topology

Although the network structures were similar (backbones and several LANs), the generation of unnecessary bandwidth (redundant probes) varied heavily among different topologies. For instance running the same simulation on two different topologies with the same number of nodes (100) generated two widely different results. The first blind vulnerability discovery simulation resulted in 63 redundant probes on the first topology. The second simulation on a topology with the same number of nodes resulted in 277 redundant probes. Running our proposed vulnerability discovery mechanism on the same topologies have resulted in 20 and 116 redundant probes respectively. Fig. 6.3 shows an example of two network topologies of 100 nodes used as part of the simulations. This indicates how sensitive the self-replicating and self-propagating algorithms are towards the network structure and how previous knowledge of the network topology would improve the performance of such algorithms.

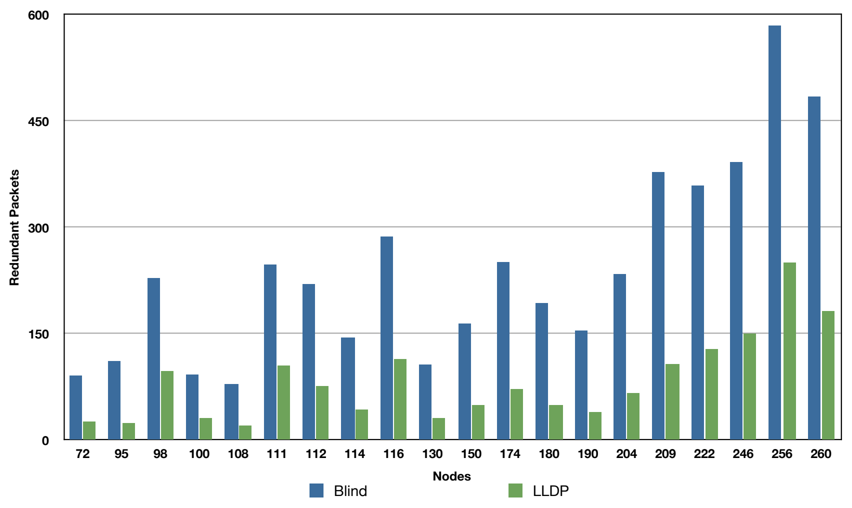


Figure 6.4: Redundant Probes.

6.9 Discussion

Even limited knowledge of network topology can reduce the communication complexity of self-replicating and self-propagating network approaches and is crucial in avoiding congestion. Vulnerability discovery cannot use highly efficient (reliable) broadcast mechanisms as it requires both interaction between entities and recovery from intermittent availability and faults since the cost of missing nodes in the discovery is disproportionate.

While blind scanning has the potential to achieve very high coverage, the number of redundant messages particularly on backbones is problematic as congestion potentially limits service availability and also restricts propagation speed. For a network of 100 nodes, our proposed mechanism reduced the number of redundant probes to 34% of the blind probing approach (on average), while for a 222-node network, the reduction was 36%, as can be seen in Fig. 6.4. Seawave does not generate redundant probes as it mainly depends on STP to propagate which avoid cycles, where there is only one

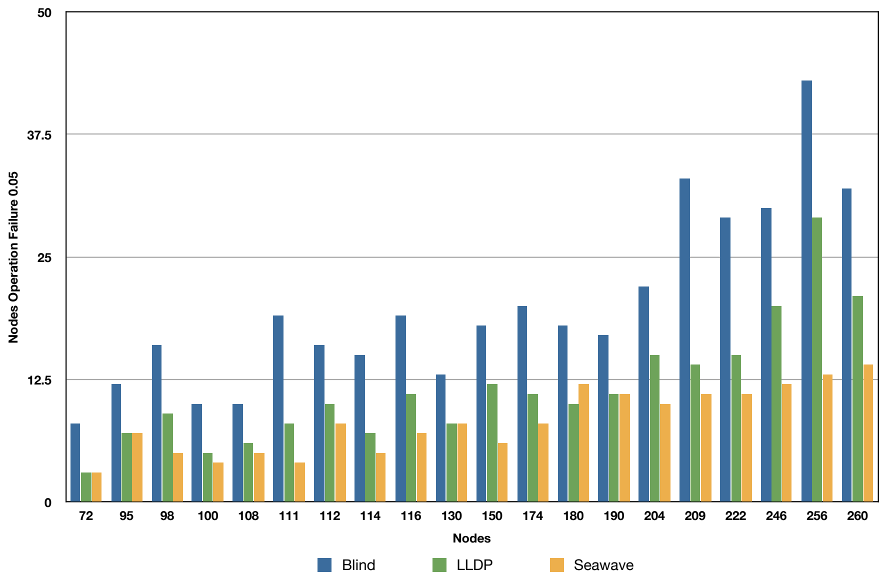


Figure 6.5: Node Operation Failure Probability of 0.05

active path to a node. However, the mechanism would generate some overhead in terms of topology information gathering, as Seawave would communicate with neighboring switches to read CAM and STP information to determine the worm propagation path and requires further recovery traffic for unavailable nodes.

The main benefit of our proposed LLDP based mechanism is the flexibility of network infrastructure supported and use of information already present on network nodes. Blind vulnerability discovery has resulted in the highest amount of link failures under the probability of $p = 0.01$ or node operation of $p = 0.05$ probability failure. That is because blind vulnerability discovery consider all links (other than the sender link) and all adjacent nodes (else sender's node) in its propagation attempts.

Our algorithm records fewer failures compared to blind vulnerability discovery as it deals with fewer links and nodes as topology information has been utilized while, Seawave has the least amount of link and node failure

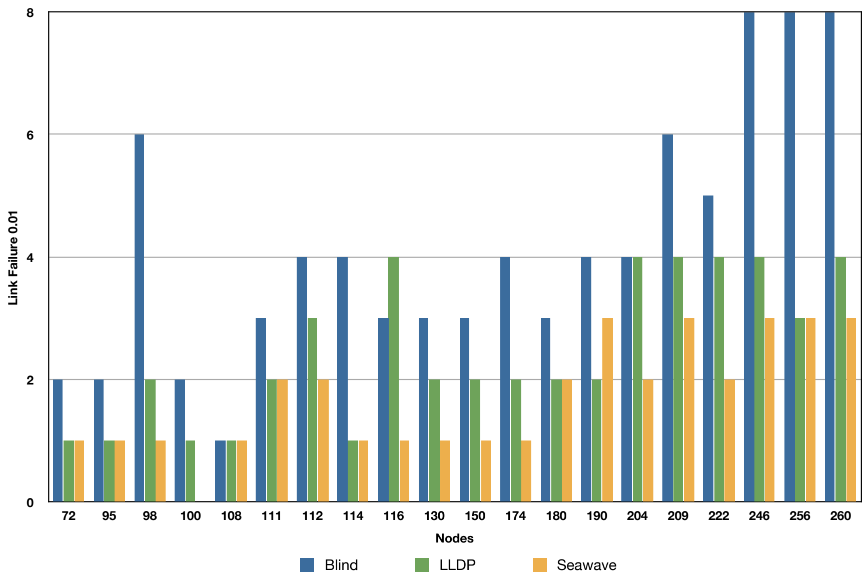


Figure 6.6: Link Failure Probability of 0.01

due to the mechanism taking advantage of STP path availability (subject to delays caused by STP convergence), as shown in Fig. 6.6 and 6.5. Results of the actual number of missed nodes during vulnerability mitigation of the three mechanisms are plotted at Fig. 6.7

6.10 Summary

Seawave I is a mechanism that depends mainly on STP information for its propagation; however, sometimes alternative approaches are needed when dealing with networks that does not necessary support STP. We have therefore, proposed in this chapter, an alternative vulnerability mitigation worm, that allocates its targets based on information retrieved by LLDP.

The use of LLDP gives the mechanism more flexibility which makes it more suitable for highly heterogeneous network architectures. We have compared the mechanism with blind vulnerability scanning and Seawave I in terms of link and node failures, redundant probes, and number of nodes

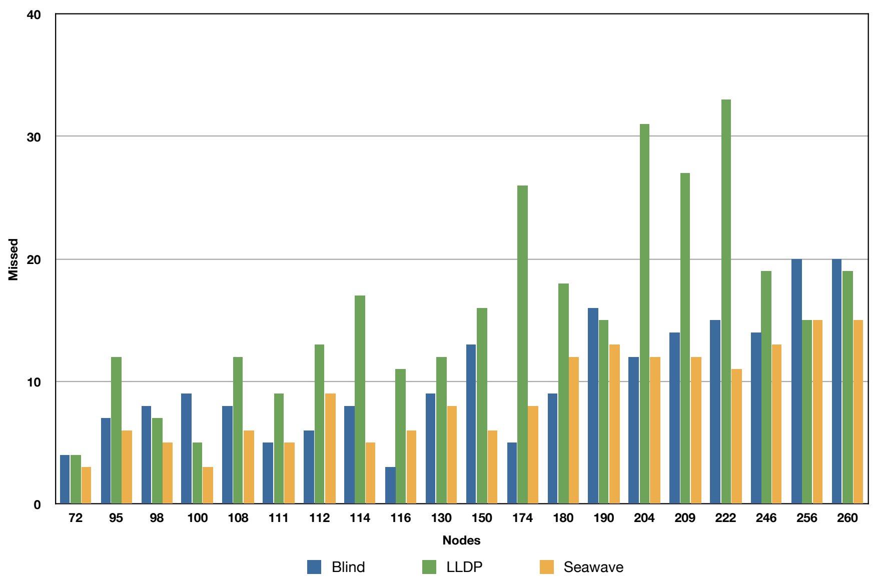


Figure 6.7: Actual Number of Missed Nodes.

the mechanisms failed to cover.

Future work would concentrate on increasing the efficiency and robustness of the mechanism, especially reducing the number of redundant probes and providing a backbone traversal algorithm along with further protective measures that keeps the LLDP mechanism safe from active adversaries.

Chapter 7

Security and Performance Aspects of Seawave

7.1 Overview

In Chapter 4 and 5, we have tested Seawave within different network topologies and also compared it to a random scanning worm – bandwidth wise. In this chapter, however, we release Seawave into the enterprise network in response to a malicious random scanning worm outbreak. We observe and evaluate Seawave's performance and report the results, comparing the defensive worm against the malicious one demonstrating that network immunity can be largely achieved despite a very limited warning interval. We also discuss mechanisms to protect Seawave against subversion and ensure the confidentiality and integrity of its communications.

7.2 Network Topology Model and Simulation Environment

Based on the network model previously described in Chapter 5 (Section 5.2), we have generated 5 hierarchical networks with nodes connected via a duplex-link where packets can flow in both directions, and for the results reported in section 7.5, the number of nodes have been selected between 100 and 500, with link bandwidth set to 100 Mbps and assuming a discovery packet size of 900 bytes. Fig. 7.1 show an example.

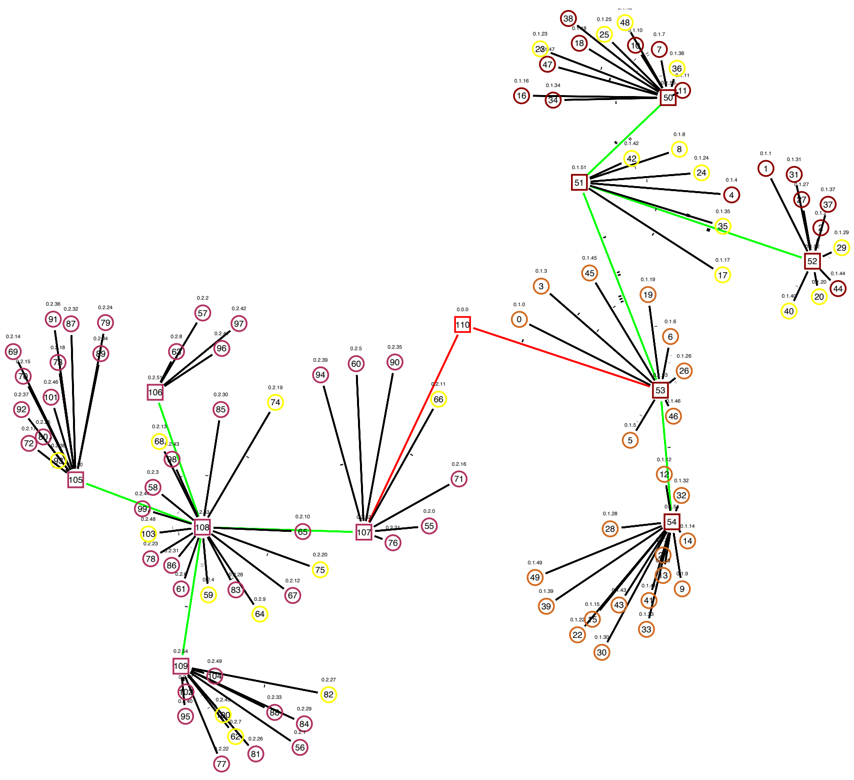


Figure 7.1: A sample of a network of 100 nodes where a malicious worm and Seawave are operating. Yellow nodes infected (by malicious worm), brown are immune (by Seawave), and red are susceptible.

7.3 Protective Measures for Seawave

Before deploying the vulnerability mitigation mechanism it is important to add protective measures that would prevent an adversary from compromising Seawave's integrity and confidentiality affecting its normal behavior. In this section we therefore, propose protective measures that should maintain the stability of Seawave during its operation and in Chapter 8 we provide a formal threat analysis model of the mechanism.

The defensive worm in its current stage does not provide sophisticated communication between plotted agents; however, it is crucial to protect the communication line between agents to block any malicious attempt to feed agents false information or even overwrite the agent code which might lead to mechanism's compromise. Different approaches exist to protect the line of communications among Seawave's agents, including:

Updating Agent Code

In some occasions, the security team of the enterprise network might need to update the agents around the network to overcome certain threats; however, many malicious users might utilize any update procedure to inject malicious code within the update to enable the compromise of Seawave. One protective measure to enable the mechanism to distribute agent updates is by using digital signatures. When Seawave needs to update its agents, it signs the update with its private key before deploying it to all agents. Recipient agents, consequently, verify the agent update by running Seawave's public key against the binary code and install the new update if it was verified successfully, otherwise, the update gets rejected. The same approach has been observed with Conficker where RSA encryption with different key lengths were used to validate or reject downloads [53]. Key distribution and management issues should be taken into account when using public key cryptography to secure agents communication, however, using it only to validate agent updates from the master node (i.e. the starting node) should not require extensive key management efforts.

Agent to Agent Communication

Agent to agent communication is needed to maintain Seawave's operational level, but there is no requirement to keep the communication between agents confidential, as it should contain only information that would keep the defensive worm running as expected (such as acknowledgment messages that an agent has self-replicated successfully). However, the integrity of agent to agent communication becomes crucial to avoid any malicious attempts to inject false information or alter packet contents to force the vulnerability mitigation mechanism to misbehave and fall out of its normal line of operation. To provide integrity and authenticity to agent to agent communication, agents can use a Message Authentication Code (MAC) algorithm, provided a secret key is hard coded in all agents. The message runs through the MAC before it is sent to its destination. The recipient agent verifies the message using the same key. For better practice the key should be changed regularly to prevent a malicious user from revealing the key by sniffing a large amount of traffic.

Agent to Master Node Communication

Since the communication between agents and the master node would contain sensitive information, bound to the sensitivity of the mission Seawave has been released on, it is crucial to maintain the confidentiality of this line of communication. Upon communication the agent suggests a symmetric key to the master node to be used as a session key, which will encrypt all the traffic between the two nodes throughout the session. The agent encrypts the session key using the master node's public key before it is sent. To prevent the malicious user from altering or suggesting the key (in other words to provide integrity and authenticity) the key is run through the MAC algorithm of the mechanism. Session keys should be generated randomly with a length that will make it hard for a malicious users to run a brute force attack against the encrypted traffic.

7.4 Switching Seawave Off

Self-replicating and self-propagating solutions should have the functionality to be switched off or un-installed from the enterprise network in case something went wrong, or even for administrative or technical requirements. Since all agents are linked to a master node, Seawave can include such functionality in different ways, including:

- The master node sends instructions to all agents to stop operating and remain dormant.

- If the agent has not received any packets from the the master node since time t then the agent stop operating and remain dormant.
- Setting a time to live (or age) for Seawave, where the whole mechanism stops operating after a pre specified time t.

The location and address of each agent should be accessible by the enterprise security team through the master node.

7.5 Simulation Results

All simulations have been performed using the Network Simulator 2 (NS-2), a discrete event simulator mainly used for research activities [42]. Following the model and assumptions outlined in section 7.2, topologies have been simulated 19 times and average results were calculated to account for random effects such as link failure probabilities and protocol state updates.

In our simulations we have gathered the following:

- The number of link failures with probability of 0.01.
- The number of node operation failures with probability of 0.05. Note that node operation failures are failures caused by the node itself (e.g. system is busy or in different state due to restarting)
- The number of packets generated by Seawave to cover the network and overcome the malicious worm. The packets that are generated between a switch and a host are, however, exempted because they have no significant impact on the bandwidth of the network. The packet size is 900 bytes.
- The time it takes Seawave to cover the corporate network and contain the malicious worm. Time is measured in simulation seconds and when there is a link or a node operation failure, the mechanism will try to recover the failure after 0.5 simulation seconds.

- The number of nodes infected by the malicious worm.
- The number of failed infection attempts due to the node being immune (i.e. entered the scope of Seawave)

Both the malicious worm and Seawave will be triggered at second 0.2 of the simulation. The malicious worm follows the model described in Chapter 4 (Section 4.6).

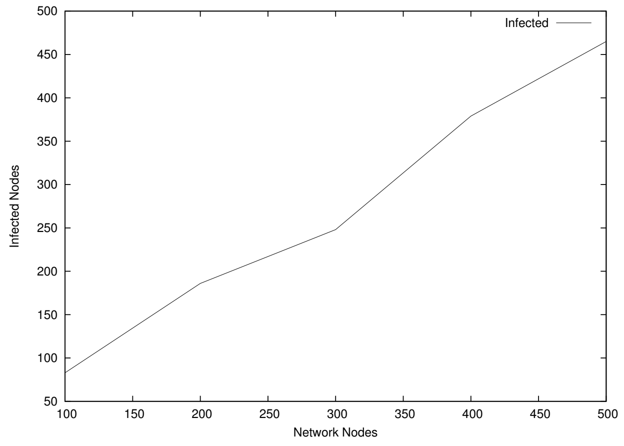


Figure 7.2: Number of nodes that the malicious random worm was able to infect, before Seawave's domination.

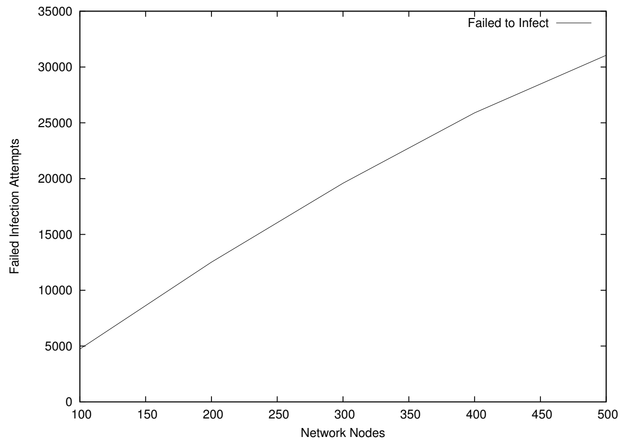


Figure 7.3: Number of failed infection attempts (triggered by the malicious worm) due to the node being immune.

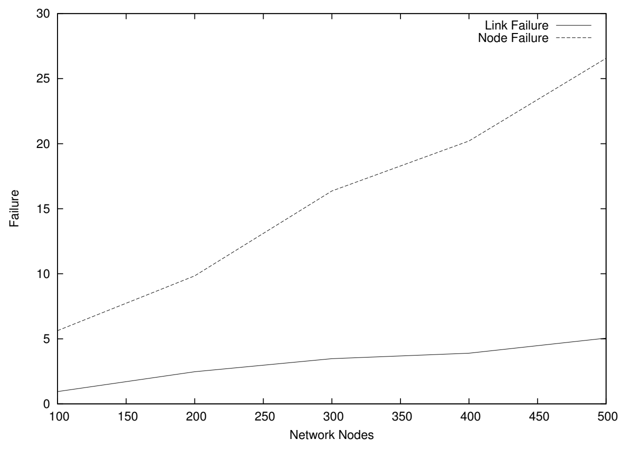


Figure 7.4: Link and Node operation failures during Seawave propagation.

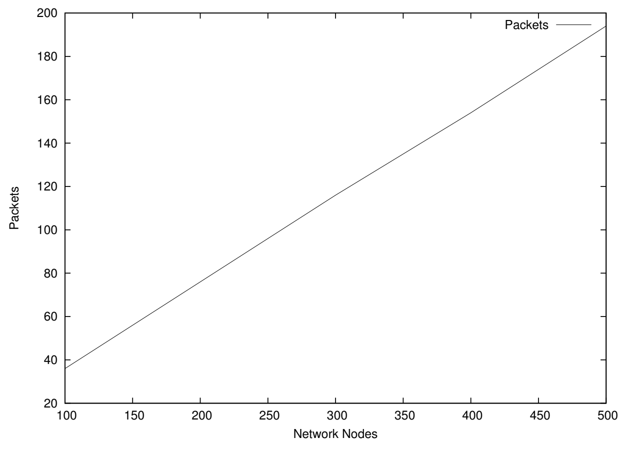


Figure 7.5: Number of packets generated by Seawave to cover the network.

7.6 Seawave Packets

During Seawave's propagation to cover the enterprise network, it communicates with other network devices using 14 types of packets. Within the switch domain, an agent communicates with the directly connected switch to detect directly connected hosts and directly connected neighboring switches. The agent then communicates with neighboring switches and hosts to propagate until it reaches the backbone where it communicates with the router to draw the backbone and propagate to other LANs. Communication between agents and the master node is also necessary for results gathering, updates,

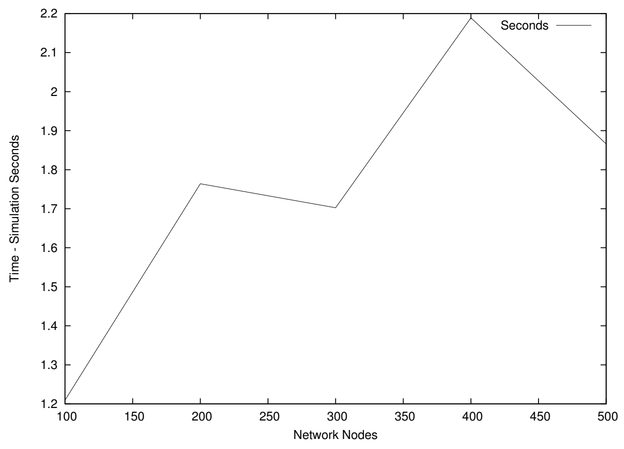


Figure 7.6: The time it required Seawave to cover the network.

and issuing further instructions. Packets classifications and descriptions utilized by the mechanism to gather topology information and propagate are summarized in Table 7.1.

7.7 Discussion

Indeed, due to the high speed of malicious worms and the lag between a vulnerability announcement and code-level patch deployment, it is necessary to have a mechanism ready to intervene when there is a significant threat to the enterprise network. The self-replicating and self-propagating nature of Seawave makes it competent to overcome malicious attacks.

When a malicious worm breaks out, it is a challenging task to eliminate it, despite the ongoing research on detection and early warning [48, 19]. With a noticeable high infection rate due to random scanning the malicious worm managed to to infect 248 nodes in a hierarchical enterprise network of 300 nodes. And with a network of 500 nodes, 465 hosts were compromised before Seawave was able to contain them. More infection results are shown in Fig. 7.2. However, the malicious worm began to encounter failed infection attempts, as some hosts have already been covered by Seawave and became immune; a network of the size of 200 nodes reported 12514

Packet Name	Description
Agent ↔ Switch	
`Seawave_STP_CAM_RQ`	Agent requests STP and CAM information from the switch.
`Seawave_STP_CAM_RPL`	Reply to agent request of STP and CAM information from the switch.
`Seawave_CAM_RQ`	Agent requests CAM information from the switches.
`Seawave_CAM_RPL`	Reply to agent request of CAM information from the switches.
Agent ↔ Host	
`Seawave_SelfReplicate`	Agent self replicates to host.
`Seawave_SelfReplicate_ACK`	Acknowledgment of agent self replication.
`Seawave_Host_Probe`	Agent probe host for vulnerability.
`Seawave_Host_Probe_ACK`	Acknowledgment of agent host probe.
Agent ↔ Router	
`Seawave_LSD_ARP_RQ`	Agent requests LSD and ARP information from router.
`Seawave_LSD_ARP_RPL`	Reply to agent request of LSD and ARP information from router.
`Seawave_ARP_RQ`	Agent requests ARP information only from router.
`Seawave_ARP_RPL`	Reply to agent request of ARP information from router.
Agent ↔ Master Node	
`Seawave_Agent_to_Master`	Agent initiates a request to the master node to start a line of communication.
`Seawave_Master_to_Agent`	Master node initiates a request to an agent to start a line of communication.

Table 7.1: Seawave's Packets Description.

failed malicious infections. And 25893 failed compromising attempts for a network of the size 400. More results are shown in Fig. 7.3

As in any mechanism, in real world deployment, the mechanism might encounter failures due to a network link or the node itself is not in a status to continue operating. For a network of the size 300 there were 16 dissemination failures due to node operation and about 26 failures for a network of 500 nodes. Network link failures have also occurred; for example 5 link failures have been observed for a network of the size 500 nodes. Obviously, the larger the size of the network the more chance that a link or node operation failure would occur. More results are shown in Fig. 7.4

Bandwidth wise Seawave was able to eliminate the malicious worm in a network of the size 300 using 116 packets. A hierarchical network that consisted of 500 nodes required 194 packets to eliminate the malicious worm and disable the vulnerability in all network nodes. More results are shown at Fig. 7.5. One of the main challenges in self-replicating and self-propagating mechanisms is maintaining the bandwidth, it has been observed that Seawave in its task to eliminate the malicious worm and cover the network has not produced high bandwidth or exhausted network main links, that is due to the fact that the defensive worm utilizes the network topology to reduce the bandwidth as much as possible.

The faster the deployment of the security mechanism the narrower the vulnerability exposure and lesser becomes the risk. The time it took Seawave to respond to the random scanning worm within a network of 300 vulnerable nodes is 1.70 simulation seconds. The time is affected of course by failures that might happen in the link or the node itself. Seawave recovers failures after 0.5 seconds. The enterprise network of 400 nodes in size has required 2.18 seconds for its coverage. Results are shown in Fig. 7.6. Note, if Seawave was not released promptly, it will be too late for the mechanism to counter the malicious worm due to network congestion.

The integrity of Seawave becomes crucial when its agents are spread around the enterprise network. Protective measures, therefore, are necessary to provide confidentiality and integrity while the mechanism operates within the network. The master node (the first node) uses public key cryptography to protect its communication with agents as described in section 7.3, however, the possibility that the private key becomes compromised by a malicious user still holds, but it is less likely as it is only one private key the security team should take care of, which narrows the scope of key management. However, the threat of compromise becomes more probable between agents as they all share the same secret key for the MAC, a malicious user who compromise an agent secret key, can communicate between agents including the master node and give misfeeds. But such a threat reduces when the agent is only receiving commands from the master nodes and not issuing them.

7.8 Summary

In this chapter we have highlighted one scenario, where a malicious worm outbreaks to an enterprise network, to be then confronted by Seawave; which has succeeded in containing the malicious worm and eliminating its threat. Using networks that mimic, to a large extent, enterprise networks, simulations of a malicious worm outbreak and Seawave attempts of containments were implemented. The results covered link and node operation failures, bandwidth generation, time of coverage, infected nodes, and failed infection attempts. In all simulation runs, Seawave was able to contain the malicious worm with minimum bandwidth.

We have also highlighted protective measures that the defensive worm can use to ensure the confidentiality and integrity of its communications. These measures include using public key cryptography to authenticate agents updates and exchange session keys with the master node; in addition to, agents sharing a secret key of the MAC algorithm to ensure the integrity of operational messages between agents.

Future work is focused on increasing both the robustness and performance of the algorithm in defending the propagation mechanism against malicious adversaries.

Chapter 8

Threat Analysis Model of Seawave Using Bayesian Belief Networks

8.1 Overview

In this chapter we propose a threat analysis model based on Bayesian Belief Networks (BBNs) to analyze and quantify threats towards Seawave. Based on several threat scenarios (Section 8.5), the model also determines the risks posed to the mechanism's duty of assessment within the enterprise network. The model constructs threats scenarios based on sequenced structure and also forms a multi-scenario threat BBN, where malicious behaviors are interdependent and threat performance aspects are adjusted using network and mechanism specific parameters. Based on predetermined event probabilities within these interdependent structures, threat scenarios likelihoods and risks are driven.

8.2 Introduction

In Chapter 4 and 5 we proposed our defensive worm Seawave and with different types of threats – especially under complex and dynamic enterprise networks – it becomes crucial to have a method to assess the mechanism's risk. Threat models allow us to prioritize the type of attacks that matters the most, mitigate Seawave's risks, discover more attacks [94] and spot the weakest points within the mechanism to allow for further enhancements. In this chapter we propose a threat analysis model to identify and quantify

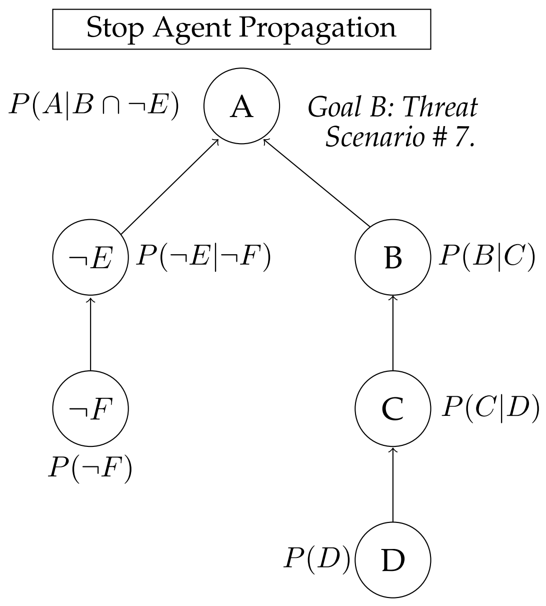


Figure 8.1: Bayesian Network of Threat Scenario 7 (See A.1.2) of Goal 8.5.2, Tree 1: Stopping the agent propagation by flooding the network with `Seawave_SelfReplicate_ACK` packets, after compromising Seawave's MAC algorithm.

threats and risks that malicious intruders might pose towards our vulnerability mitigation mechanism. The model is based on Bayesian Belief Networks (BBNs), which allows us to compute probabilistic inference of threats that might target our mechanism. BBNs have four distinctive capabilities as illustrated by Heckerman [39] and a fifth capability added by Pendharkar, *et al.* [78]:

- Handling incomplete data sets,
- Learning casual relationships,
- Combining domain (prior) knowledge and data,
- Avoiding the over fitting of data, and
- Ability of updating the probability distribution.

These capabilities provide efficient estimates that lead to sufficient managerial decision making based on previous knowledge and subjective probabilistic predictions.

8.3 Related Work

Different models to analyze applications from different perspectives use Bayesian Belief Networks since they provide a robust inductive reasoning. Chulani *et al.* demonstrated few limitations of multiple regression cost models and how a more sophisticated Bayesian approach would overcome these limitations. The authors move on to compare and contrast the two models and conclude that the Bayesian approach is more accurate and robust than the multiple regression approach [20]. Likewise, Fenton *et al.* show how BBNs can support effective risk management decisions compared to traditional metrics approaches [28, 27], they use a decision-support toolset to validate their proposal.

Pendharkar *et al.* proposed a Bayesian model and compared it with other approaches such as neural network and regression tree. Their results show how BBNs are more competitive in generating point forecasts, how probabilistic bounds can be set by managers on software effort forecasts, and how new subjective estimates can be incorporated to the Bayesian model [78]. Although, there is a reasonable amount of literature review on different use of BBNs in assessing software, it is not easy to allocate a research on risk assessment methods based on BBNs to identify and analyze casual threats as also noted by [52].

Foroughi described an intelligent agent that uses bayesian techniques to learn prior risks factors and asset properties in order to generate point factors and also adjust its probability distribution based on new results [32], while Phillips *et al.* proposed a probabilistic graph-based approach to network vulnerability analysis, which can also analyze risks to network assets [79] . Kondakci proposed a BBN based causal risk assessment method (CRAM) to identify and analyze threats and quantify risks associated with them. CRAM can be used to conduct inductive and deductive reasoning [52].

8.4 Threat Model Components

Indeed threats come in different types and from various sources, it becomes necessary to classify threats according to their main objective. Here we divide the threats towards our vulnerability mitigation worm into eight components according to the goals adversaries try to achieve:

- ***A: Malicious Use of Agent to Master Node Communication.*** Under this component comes the threats intervening with the communication line between an agent and the mechanism's master node.
- ***B: Malicious Use of Agent to Agent Communication.*** The malicious interventions in agent to another agent line of communication are included within this component.
- ***C: Compromise Agent to Switch Communication.*** The agent communicates with the switch to retrieve important topology information. The threats to that line of communication are included under this component.
- ***D: Compromise Agent to Host Communication.*** When an agent communicates with a host node or vice versa, malicious abuse of this communication line is included under this component.
- ***E: Compromise Agent to Router Communication.*** When the agent reads router information for further propagation in the enterprise network, such communication is exposed to different threats included under this component.

- ***F: Unauthorized Modification of Agent Code.*** This component describes malicious attempts to abuse the code update process within the mechanism.
- ***G: Compromising Agent in Host Machine.*** Attack sequences towards compromising the agent resident in a network node are included under this component.
- ***H: Mechanism Information Gathering.*** Attempts to gather information about the mechanism for malicious use are described under this component.

Next we elaborate on the threat interdependent structure within each component.

8.5 Mechanism's Threat based Bayesian Networks

8.5.1 Goal: Malicious Use of Agent to Master Node Communication

1. A: Decrypt the Communication Line Between Master Node and Agent.

   1.1 B: Break Asymmetric Encryption.

      1.1.1 C: Break Asymmetric Encryption by Brute-force.

      1.1.2 D: Mathematically Break Asymmetric Encryption.

2. G: Exhaust Master Node Memory (DoS).

   2.1 H: Request Connections to Master Node Based on Reply-Attacks.

      2.1.1 I: Sniff Network Traffic for `Seawave_Agent_to_Master` packets.

         2.1.1.1 J: Compromise a Network Node.

         2.1.1.2 N: Plug into the Network.

   2.2 K: SYN Flood

3. L: Feed Malformed Information to Master Node.

   3.1 M: Compromise an Agent Node.

8.5.2 Goal: Malicious Use of Agent to Agent Communication

1. A: Stop Agent Propagation - Abnormally.

   1.1 B: Send a fake `Seawave_SelfReplicate_ACK` packet to the agent (Reply-Attack).

      1.1.1 C: Flood LAN with unauthentic `Seawave_SelfReplicate_ACK` packets.

         1.1.1.1 D: Compromise the MAC algorithm.

   1.2 E: Send back an authentic `Seawave_SelfReplicate_ACK` packet.

      1.2.1 F: Compromise the Selected Vulnerable Node.

2. G: Compromising the MAC algorithm.

   2.1 H: Brute-force MAC Key.

   2.2 I: Compromise an Agent Node.

3. J: Stop Agent propagation to Next LAN.

   3.1 P: Send fake `Seawave_SelfReplicate_ACK` to Agent.

      3.1.1 K: ARP Cache Poisoning of Router to Point to Compromised Node.

         3.1.1.1 L: Compromise Host Node at Next LAN.

4. M: Capture Agent Code after Self-Replicating to an already Compromised Node.

   4.1 N: Flood CAM table with Compromised Host MAC Address.

      4.1.1 O: Compromise Host Node.

      4.1.2 Q: Plug intruder machine into the target switch.

8.5.3 Goal: Compromise Agent to Switch Communication

1. A: Redirect Agent Vulnerability Probing to a Compromised Host.
   - 1.1 B: Flood CAM table with MAC Address of rouge host.
     - 1.1.1 I: Compromise Host Node.
     - 1.1.2 J: Plug intruder machine to target switch.
2. C: Generate Denial of Service (DoS) attack.
   - 2.1 D: Form a Loop by Manipulating STP Next Bridge field
3. E: Stop Propagation - Abnormally.
   - 3.1 F: Change STP Next Bridge field to a non existent Address and Flood CAM table.
4. G: Redirect Mechanism to Next LAN.
   - 4.1 H: Change STP Next Bridge field to point to Router.

8.5.4 Goal: Compromise Agent to Host Communication

1. A: Capturing Exploit code used by Agent to detect Vulnerability.
   - 1.1 B: Compromise Host Machine.
   - 1.2 H: Plug Rouge Host Machine to target Switch.
2. C: Stop Mechanism from Probing Hosts for Vulnerabilities.
   - 2.1 D: Flood hosts with `Seawave_Host_Probe_ACK` packets.
     - 2.1.1 I: Plug Rouge Host Machine to target Switch.
3. E: Stop Mechanism from Self-Replicating to next Switch.
   - 3.1 F: Flood hosts with `Seawave_SelfReplicate_ACK` packets.
   - 3.2 G: Send a `Seawave_SelfReplicate_ACK` unicast packet to the Agent.

8.5.5 Goal: Compromise Agent to Router Communication

1. A: Stop Mechanism from Propagating or Redirect The propagation.
   - 1.1 B: Feed Malformed OSPF (Link State Database) to Mechanism.
     - 1.1.1 C: Manipulate OSPF Data.
   - 1.2 D: Give Corrupted ARP Table.
     - 1.2.1 E: Manipulate SNMP Data.
2. F: Impersonate a Router.
   - 2.1 I: Modify Switch Next Bridge to Point to impersonator host machine.
     - 2.1.1 J: Plug impersonator machine to target switch.

8.5.6 Goal: Unauthorized Modification Agent Code

1. A: Compromise Master node Private Key.
   - 1.1 B: Compromise Master Node Machine.
   - 1.2 C: Brute force Private Key.

8.5.7 Goal: Compromising Agent in Host Machine

1. A: Exploiting (root-privilege) Vulnerability in Host Machine
   - 1.1 B: Vulnerability Scanning of target system.
     - 1.1.1 C: Port Scanning Target.

8.5.8 Goal: Mechanism Information Gathering

1. A: Store Mechanism Communication Activity.
   - 1.1 B: Start sniffing agent traffic.
     - 1.1.1 C: When probed record agent IP/MAC.

8.6 Multiple Adversaries Bayesian Belief Threat Network

This section describes the probability of attacks based on more than one attacker.

8.6.1 Goal: Halt Mechanism's Propagation

1. S_{21}: Stop Mechanism From Propagating Over the Backbone.

   1.1 S_{27}: Information Gathering of Mechanism activity within the Enterprise Network.

2. S_{15}: Stop Agent from propagating Beyond the current switch domain.

   2.1 S_{27}: Information Gathering of Mechanism activity within the Enterprise Network.

8.7 Threat BBN Conditional Probabilities

8.7.1 Goal: Malicious Use of Agent to Master Node Communication

1. $A : P(A|B) = 0.1, P(A|\neg B) = 0.6$

   1.1 $B : P(B|C \cap D) = 0.01, P(B|C \cap \neg D) = 0.2, P(B|\neg C \cap D) = 0.01, P(B|\neg C \cap \neg D) = 0.5$

   1.1.1 $C : P(C) = 0.3$

   1.1.2 $D : P(D) = 0.001$

2. $G : P(G|H \cap K) = 0.6, P(G|\neg H \cap K) = 0.8, P(G|H \cap \neg K) = 0.5, P(G|\neg H \cap \neg K) = 0.2$

   2.1 $H : P(H|I) = 0.4, P(H|\neg I) = 0.5$

   2.1.1 $I : P(I|J \cap N) = 0.2, P(I|J \cap \neg N) = 0.3, P(I|\neg J \cap N) = 0.5, P(I|\neg J \cap \neg N) = 0.1$

   2.1.1.1 $J : P(J) = 0.3$

2.1.1.2 $N : P(N) = 0.5$

2.2 $K : P(K) = 0.7$

3. L: $P(L|M) = 0.7, P(L|\neg M) = 0.1$

3.1 $M : P(M) = 0.2$

8.7.2 Goal: Malicious Use of Agent to Agent Communication

1. $A : P(A|B \cap E) = 0.3, P(A|B \cap \neg E) = 0.7, P(A|\neg B \cap E) = 0.2, P(A|\neg B \cap \neg E) = 0.5$

1.1 $B : P(B|C) = 0.5, P(B|\neg C) = 0.3$

1.1.1 $C : P(C|D) = 0.3, P(C|\neg D) = 0.5$

1.1.1.1 $D : P(D) = 0.1$

1.2 $E : P(E|F) = 0.3, P(E|\neg F) = 0.005$

1.2.1 $F : P(F) = 0.2$

2. $G : P(G|H \cap I) = 0.2, P(G|H \cap \neg I) = 0.2, P(G|\neg H \cap I) = 0.2, P(G|\neg H \cap \neg I) = 0.2$

2.1 $H : P(H) = 0.2$

2.2 $I : P(I) = 0.2$

3. $J : P(J|P) = 0.6, P(J|\neg P) = 0.2$

3.1 $P : P(P|K) = 0.5, P(P|\neg K) = 0.4$

3.1.1 $K : P(K|L) = 0.2, P(K|\neg L) = 0.4$

3.1.1.1 $L : P(L) = 0.2$

4. $M : P(M|N) = 0.5, P(M|\neg N) = 0.2$

4.1 $N : P(N|O \cap \neg Q) = 0.5, P(N|\neg O \cap Q) = 0.5$

4.1.1 $O : P(O) = 0.2, P(\neg O) = 0.8$

4.1.2 $Q : P(Q) = 0.3$

8.7.3 Goal: Compromise Agent to Switch Communication

1. $A : P(A|B) = 0.6, P(A|\neg B) = 0.2$

   1.1 $B : P(B|I \cap \neg J) = 0.5, P(B|\neg I \cap J) = 0.5$

      1.1.1 $I : P(I) = 0.2$

      1.1.2 $J : P(J) = 0.3$

2. $C : P(C|D) = 0.6, P(C|\neg D) = 0.3$

   2.1 $D : P(D) = 0.7$

3. $E : P(E|F) = 0.6, P(E|\neg F) = 0.4$

   3.1 $F : P(F) = 0.7$

4. $G : P(G|H) = 0.6, P(G|\neg H) = 0.3$

   4.1 $H : P(H) = 0.7$

8.7.4 Goal: Compromise Agent to Host Communication

1. $A : P(A|B \cap H) = 0.5, P(A|\neg B \cap H) = 0.5, P(A|B \cap \neg H) = 0.5, P(A|\neg B \cap \neg H) = 0.2$

   1.1 $B : P(B) = 0.2$

   1.2 $H : P(H) = 0.3$

2. $C : P(C|D) = 0.5, P(C|\neg D) = 0.2$

   2.1 $D : P(D|I) = 0.5$

      2.1.1 $I : P(I) = 0.3$

3. $E : P(E|F \cap G) = 0.4, P(E|F \cap \neg G) = 0.5, P(E|\neg F \cap G) = 0.4, P(E|\neg F \cap \neg G) = 0.3$

   3.1 $F : P(F) = 0.5$

   3.2 $G : P(G) = 0.5$

8.7.5 Goal: Compromise Agent to Router Communication

1. $A : P(A|B \cap D) = 0.3, P(A|B \neg D) = 0.5, P(A|\neg B \cap D) = 0.5, P(A|\neg B \cap \neg D) = 0.3$

   1.1 $B : P(B|C) = 0.4, P(B|\neg C) = 0.3$

   1.1.1 $C : P(C) = 0.4$

   1.2 $D : P(D|E) = 0.5, P(D|\neg E) = 0.3$

   1.2.1 $E : P(E) = 0.4$

2. $F : P(F|I) = 0.5, P(F|\neg I) = 0.3$

   2.1 $I : P(I|J) = 0.5, P(I|\neg J) = 0.3$

   2.1.1 $J : P(J) = 0.6$

8.7.6 Goal: Unauthorized Modification Agent Code

1. $A : P(A|B \cap C) = 0.1, P(A|\neg B \cap C) = 0.2, P(A|B \cap \neg C) = 0.5, P(A|\neg B \cap \neg C) = 0.3$

   1.1 $B : P(B) = 0.3$

   1.2 $C : P(C) = 0.1$

8.7.7 Goal: Compromising Agent in Host Machine

1. $A : P(A|B) = 0.4, P(A|\neg B) = 0.1$

   1.1 $B : P(B|C) = 0.5, P(B|\neg C) = 0.3$

   1.1.1 $C : P(C) = 0.5$

8.7.8 Goal: Mechanism Information Gathering

1. $A : P(A|B) = 0.7, P(A|\neg B) = 0.1$

   1.1 $B : P(B|C) = 0.5, P(B|\neg C) = 0.3$

   1.1.1 $C : P(C) = 0.7$

8.8 Multiple Adversaries BBN Conditional Table

8.8.1 Goal: Halt Mechanism's Propagation

1. $S_{21} : P(S_{21}|S_{27}) = 0.4, P(S_{21}|\neg S_{27}) = 0.2$

   1.1 $S_{27} : P(S_{27}) = 0.147$

2. $S_{15} : P(S_{15}|S_{27}) = 0.6, P(S_{15}|\neg S_{27}) = 0.4$

   1.1 $S_{27} : P(S_{27}) = 0.147$

8.9 Attack Scenarios

In this section we will highlight possible attack scenarios that would affect Seawave's performance and try to find the mechanism's weakest points. In goal 8.5.1 where the attacker tries to compromise the communication line between the agent and the master node, scenario 1 (See A.1.1) calculates the possibility of an attacker decrypting the communication line by breaking the RSA encryption mathematically. However, the lack of practical relevance of such attacks make them less likely to occur.

In scenario 2 (See A.1.1), the attacker tries to decrypt agent communication to master node by exhaustive key search, which is also very less likely to succeed as it requires high computing power that is not available to average attackers. Both scenarios resulted in $7 \times 10^{-5}\%$ and 0.5994% probabilities respectively. However, in scenario 3 (See A.1.1) Seawave might be weakened if the adversary was able to connect to the enterprise network and sniff agent to master node communication request packets and thereafter use them to initiate large amount of `Seawave_Agent_to_Master` requests forming a DoS against the master node, limiting its availability. The probability computed for the attack was 1.05%. Conducting the same scenario, but using an already compromised network node as in scenario 4 (See A.1.1) would result in the probability decreasing to 0.27%, that is due to the difficulty of compromising a node compared to plugging the adversary node to the network before initiating the attack. Flooding the master node with SYN connections has resulted in a probability of 8.82% as computed in scenario 5 (See A.1.1). In an attempt to feed the master node with malformed data as in scenario 6 (See A.1.1), the adversary compromises an agent node and starts a line of malformed communication with the master node, such probability was calculated as 14%.

In the context of malicious use of agent to agent communication 8.5.2, scenario 7 (See A.1.2) describes the possibility of an adversary succeeding in stopping Seawave's propagation by flooding the network with `Seawave_SelfReplicate_ACK` packets in hope that agents about to propagate receive the packet and stop. However, such attack require the attacker to compromise the MAC algorithm and learn the secret key, which is not straightforward; the probability was computed to be 0.8358%.

Scenario 8 (See A.1.2), stops the mechanism by sending an authentic `Seawave_SelfReplicate_ACK` packet to the source agent after the adversary succeeds in compromising the vulnerable node. The intruder needs to pick the same node, randomly selected by Seawave, which is very unlikely; the probability calculated for such scenario was 0.378%. The probability of compromising the MAC algorithm used by Seawave using brute force attack was 3.2% as computed in scenario 9 (See A.1.2) and by compromising the agent and thereafter extracting the secret key it has resulted to 12.8% as in scenario 10 (See A.1.2).

The agent can be stopped from propagating to other LANs in the enterprise by compromising a host in the LAN and ARP cache poisoning the router to point to the compromised node. When the agent fetches the ARP cache of the router it will pick the intruder node and therefore self-replicate to that already compromised node. The intruder, thereafter, will send back a `Seawave_SelfReplicate_ACK` packet to refrain the agent from propagating further. Scenario 11 (See A.1.2) covered this attack with a probability of 1.2%. The intruder after compromising a node, can trick the agent to pick the mac address of the malicious node by flooding the switch CAM table with the same MAC as the intruder's. The attacker can, thereafter, capture the agent code of the mechanism (during self-replicating) as in scenario 12 (See A.1.2), which resulted in a probability of 3.5%.

The intruder might be able to compromise agent to switch line of communication 8.5.3. In scenario 13 (See A.1.3) the intruder can redirect the agent to probe a rouge host for vulnerability by compromising a host node and flooding the switch with the MAC address of the rouge node. The probability of such attack results in 4.2%. Another malicious interaction with the switch might result in a DoS attack on the mechanism by altering the STP next bridge field in the switch to point to another switch forming a loop. The probability of the attack was calculated in scenario 14 (See A.1.3) to be 42%. The intruder can stop the mechanism by altering STP next bridge to point to a non-existent switch and flooding the CAM with fake MAC addresses to remove all switch addresses, such attack probability is 42% as in scenario 15 (See A.1.3). Seawave can be directed to shift to the next LAN without covering all its current LAN by altering the next bridge field in the switch to point to the router, forcing the agent to start traversing the enterprise network. However, the mechanism will still propagate throughout switches that exist in the CAM table. Scenario 16 (See A.1.3) highlight this attack with a probability of 42%.

Within the switch domain, where the agent communicates with host nodes connected to the same switch, malicious activities are expected 8.5.4. An intruder compromising a machine can capture the exploit used by the agent upon probing the vulnerability, however, in general vulnerabilities addressed by the mechanism would most likely be publicly available exploits, reducing the impact of the attack. The probability of such activity resulted in 7% as calculated in scenario 17 (See A.1.4). The intruder might be able to stop the mechanism from probing host nodes connected to the same switch by flooding all nodes within the switch domain with `Seawave_Host_Probe_ACK` packets causing the agent to consider host nodes already probed, while they are not. Probability of this malicious activity occurring is 7.5% as in scenario 18 (See A.1.4). Inline with the previous technique the attacker can also stop the agent from propagating to neighboring switches by flooding host nodes with a `Seawave_SelfReplicate_ACK` packet to mislead the agent that it has already self-replicated to next switches. The possibility of scenario 19 (See A.1.4) was 12.5%. The same scenario can also occur without flooding by sending a unicast `Seawave_SelfReplicate_ACK` packet to the agent directly, but it requires the intruder to successfully allocate the agent. The attack probability was 0.5% as in scenario 20 (See A.1.4).

Agent to router communication has its share of malicious activity 8.5.5, as an intruder can stop the mechanism from traversing the backbone by altering the LSD OSPF data (`Seawave_LSD_ARP_RPL`), e.g. pointing to no further routers, before it reaches the agent. As in scenario 21 (See A.1.5) the probability was 2.4%. Manipulating the ARP table of the router (`Seawave_ARP_RPL`) would also cause the intruder to redirect the agent to whatever IP address provided or even stopping the agent from spreading to other LANs connected to the same router. This attack probability was 3.6% as in scenario 22 (See A.1.5). The intruder can also impersonate a router by modifying STP next bridge field to point to the malicious node which will act as a router to provide misleading information to the mechanism. The probability computed was 15% as in scenario 23 (See A.1.5).

In order for the intruder to be able to carry unauthorized modifications of agent code (8.5.6), he has to locate then compromise the master node, before exposing the private key, which is used to sign the agent code. The probability of such attack results in 13.5% as in scenario 24 (See A.1.6). Revealing the private key by exhaustive key search, consumes long time and require high computing resources, the likelihood of the attack was 1.4% as in scenario 25 (See A.1.6).

For the intruder to compromise the agent, as in goal 8.5.7, he has to secure access to the network and locate the agent node before starting port scanning to identify running network applications on the agent machine. Then the attacker starts probing his target for vulnerabilities, if there was an exploitable vulnerability that would grant the intruder with root privilege then go ahead and exploit to compromise the machine and thereafter the agent. The probability of the attack was 10% as in scenario 26 (See A.1.7).

Indeed information gathering of a target is one of the crucial elements of a successful attack. Goal 8.5.8 cover this area, where the likelihood of an intruder accessing the network and allocating the agent at the current switch – upon vulnerability probing – before recording and analyzing agent traffic to allocate the mechanism's master node is described in scenario 27 (See A.1.8) and has resulted in a probability of 24.5%.

8.10 Seawave's Threat Model

The approach we have used to assess the security of Seawave is based on Bayesian Belief Networks. BBN helps us calculate threats and their conditional dependencies, forming a threat scenario that targets the mechanism. Each threat consists of sub-threats that forms a tree of threats to achieve a certain malicious goal against the mechanism. These bayesian nets (or trees) are constructed as a directed acyclic graph (DAG) and each node in the graph, lists the conditional probability table based on the parent node probability table. Each node can be described as conditionally independent given its parent nodes. Fig. 8.1 shows a threat scenario BBN (scenario 7, see A.1.2) that might be launched against Seawave. Threat probabilistic inference can be drawn for different attack techniques when conditional probability tables are filled for each node within the threat BBN. By rearranging the conditional probability formula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

We get the chain rule, where we can calculate the probability of threat A taking into account its dependencies:

$$P(A \cap B) = P(A|B)P(B) \tag{8.1}$$

Thereafter, by symmetry we get the well known Bayes' rule:

$$P(A|B) = \frac{P(B \cap A)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \tag{8.2}$$

The notation $P(A \cap B)$ describes the joint probability of threat A and threat B and the notation $P(A|B) = p$ means given threat B and everything else is irrelevant to threat A then the probability of A is p [44]. The Baye's rule allows us to revisit our estimations of threat A given that we get information about another threat B. The denominator refers to the marginal (unconditional) probability of event B; that is B regardless of other events, which according to the law of total probability can be computed by:

$$P(B) = \sum_{i=1}^{n} P(B \cap A_i) = \sum_{i=1}^{n} P(A_i)P(B|A_i) \tag{8.3}$$

In order to compute results, each node in the BBN (or threat tree) should have a conditional probability table. We have created that table based on subjective probability driven from work experience and current state of information. Conditional probabilities of each node are driven based on the value of their parent nodes. Suppose the set of threats in a BBN is $\{T_1, T_2, \ldots, T_n\}$ and $Parents(T_i)$ denote the set of parents of the node T_i in the same threat BBN, then the joint probability distribution of the BBN can be calculated by:

$$P(T_1 \cap T_2 \cap T_3 \cap \ldots T_n) = \prod_{i=1}^{n} P(T_i|Parents(T_i)) \tag{8.4}$$

For example, from Eq. (8.4) we can compute the the probability of Fig. 8.1 that is: *Stopping the agent propagation by flooding the network with `Seawave_SelfReplicate_ACK` packets after compromising the MAC algorithm*, by calculating the joint probability of the threat BBN:

$$P(A \cap B \cap C \cap D \cap \neg E \cap \neg F) =$$
$$P(A|B \cap \neg E) \cdot P(B|C) \cdot P(C|D) \cdot P(D) \cdot P(\neg E|\neg F) \cdot P(\neg F)$$

In Appendix A we have constructed several threat scenarios that might pose harm to Seawave. These threats are grouped to achieve certain malicious tasks under pre specified goals (see section 8.4). The conditional tables of these threat networks are described in section 8.7, where we can derive probabilistic inference for each BBN. For example, if we observe threat scenario 7 (See A.1.2) we calculate the probability of this threat taking part to be 0.8358%, this result is based on providing evidence (findings) for each node in the threat BB network. But what if we are interested in computing the probability of threat B taking place within the network regardless to other threats (i.e. unconditional probability). From Eq. (8.3) to calculate the marginal probability of threat B ($P(B)$) (8.5.2) that is the probability of a fake `Seawave_SelfReplicate_ACK` packet reaching the mechanism's agent, we first calculate $P(C)$ based on the conditional table 8.7.2.

$$P(C) = P(C|D)P(D) + P(C|\neg D)P(\neg D) = (0.3 \cdot 0.1) + (0.5 \cdot 0.9) = 0.48$$

Note D is a root node (has no parents) and therefore P(D) equals the conditional probability in the table. Now we are ready to calculate $P(B)$:

$$P(B) = P(B|C)P(C) + P(B|\neg C)P(\neg C) = (0.5 \cdot 0.48) + (0.3 \cdot 0.52) = 0.396$$

Therefore, the unconditional probability of threat B taking place is 39.6%.

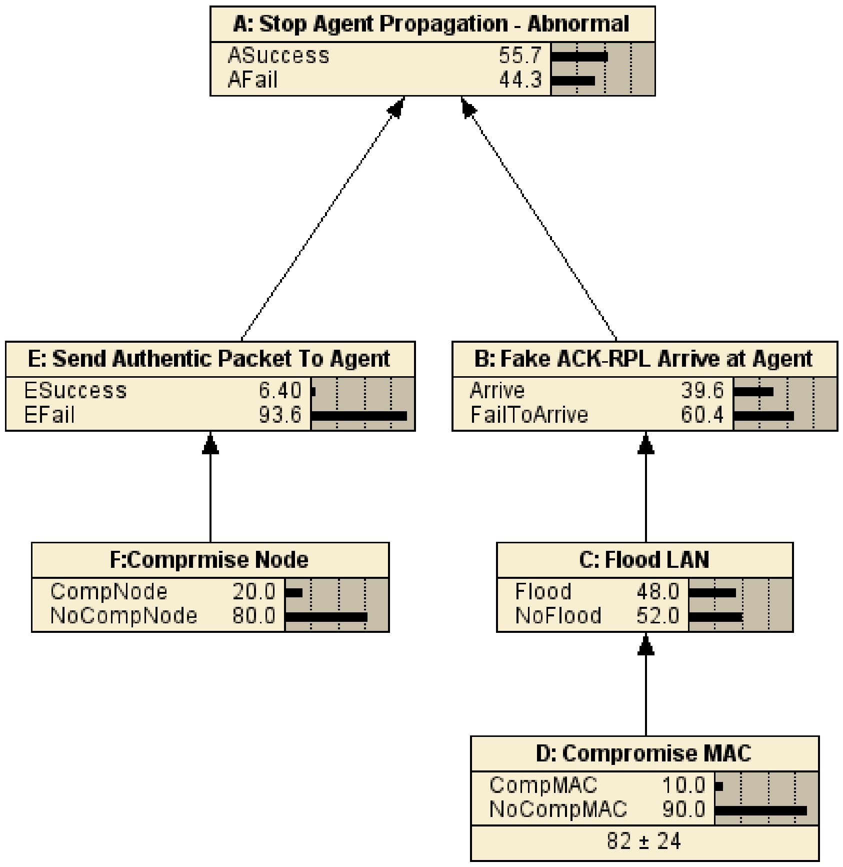


Figure 8.2: Seawave Bayesian Belief Threat Network 8.5.2 (Tree 1) without providing any evidence (Netica screenshot).

Fig. 8.2 shows event A (8.5.2) BBN without entering any evidence. However, one of the benefits of BBNs is the ability to revisit the probabilities upon new evidence. For example, suppose that we do not know that there was any flooding of an unauthentic `Seawave_SelfReplicate_ACK` packets within the LAN (event C 8.5.2), but we do know that the agent has received a forged `Seawave_SelfReplicate_ACK` packet (event B 8.5.2). Providing the evidence that event B is true, then using Bayesian theorem Eq. (8.2), we can determine the revised probability that there was flooding:

$$P(C|B) = \frac{P(B|C)P(C)}{P(B)} = \frac{0.5 \cdot 0.48}{0.396} = 0.61$$

Therefore the observation that the agent has actually received a forged `Seawave_SelfReplicate_ACK` packet (event B is true), has increased the probability that the LAN has been flooded by an unauthentic `Seawave_SelfReplicate_ACK` packets (event C) up from 0.48 to 0.61.

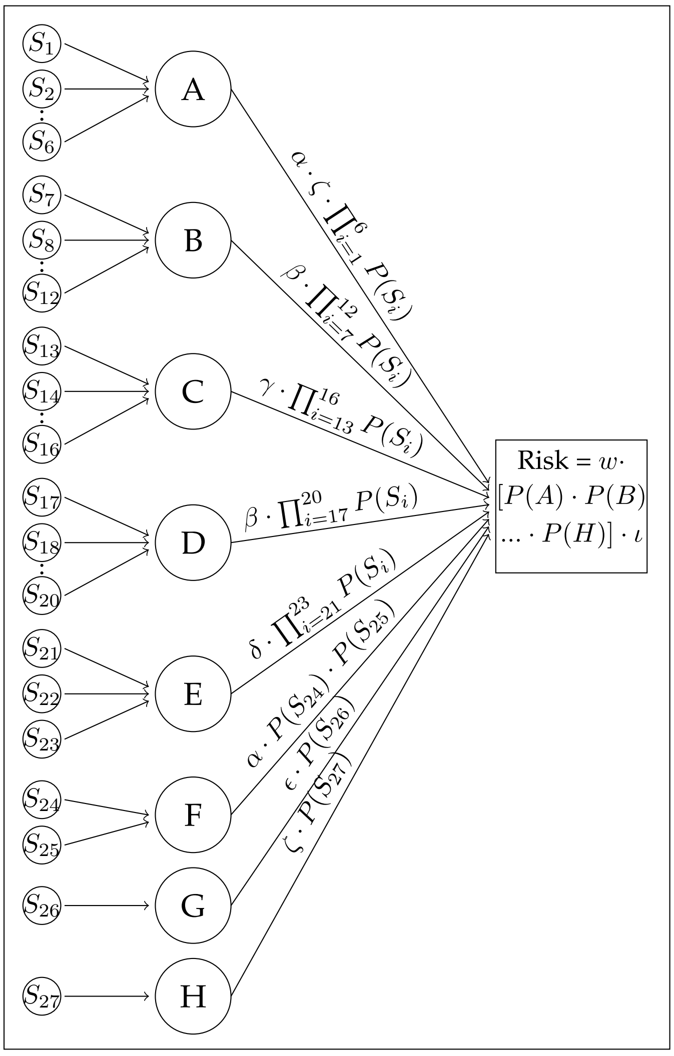

S_1
S_2
S_6
S_7
S_8
S_{12}
S_{13}
S_{14}
S_{16}
S_{17}
S_{18}
S_{20}
S_{21}
S_{22}
S_{23}
S_{24}
S_{25}
S_{26}
S_{27}
A
B
C
D
E
F
G
H
$\alpha \cdot \zeta \cdot \prod_{i=1}^{6} P(S_i)$
$\beta \cdot \prod_{i=7}^{12} P(S_i)$
$\gamma \cdot \prod_{i=13}^{16} P(S_i)$
$\beta \cdot \prod_{i=17}^{20} P(S_i)$
$\delta \cdot \prod_{i=21}^{23} P(S_i)$
$\alpha \cdot P(S_{24}) \cdot P(S_{25})$
$\epsilon \cdot P(S_{26})$
$\zeta \cdot P(S_{27})$
Risk = $w \cdot$
$[P(A) \cdot P(B)$
$... \cdot P(H)] \cdot \iota$

Figure 8.3: Threat Analysis Model of Seawave.

8.10.1 Constructing The Threat Analysis Model

Based on this we construct a threat analysis model to determine the security risks towards Seawave as shown at Fig. 8.3. The model divides threat scenarios into different groups and applies certain mechanism or network parameters to address performance aspects of the malicious attack. These parameters adjust the ability of the malicious attacker to:

- Allocate the Master Node: α
- Recognize and generate Seawave's traffic: β
- Manipulate STP and CAM topology information: γ
- Manipulate OSPF and ARP topology information: δ
- Allocate Seawave's Agent: ϵ
- Sniff traffic in a switched network: ζ
- Secure access to enterprise network (global parameter): ι

In the model we consider each threat scenario as an independent event, that is $P(S_1|S_2) = P(S_1)$. We, therefore, calculate the joint probability of each goal using the multiplication rule:

$$P(S_1 \cap S_2) = P(S_1) \cdot P(S_2)$$

The parameter(s) p are then considered to adjust attack performance, as follows:

$$p \cdot \prod_{i=0}^{n} P(S_i);\ P(S_i) \in \{A, B...H\}, p \in \{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\} \tag{8.5}$$

The parameter ι which addresses the possibility of an attacker securing access to the enterprise network is a global parameter and therefore applies to all attack scenarios. To see how these threat scenarios affects Seawave, we define a weight and risk value for the asset targeted, inspired by [52] with slight modification. We assume asset weight value w range from 0 to 3 and risk value $0 \leq \mathrm{R} \leq 3$ to identify different risk levels: Low {0.0 - 1.0}, Medium {1.1 - 2.0}, and High {2.1 - 3.0}. The risk to the vulnerability mitigation mechanism is therefore calculated by:

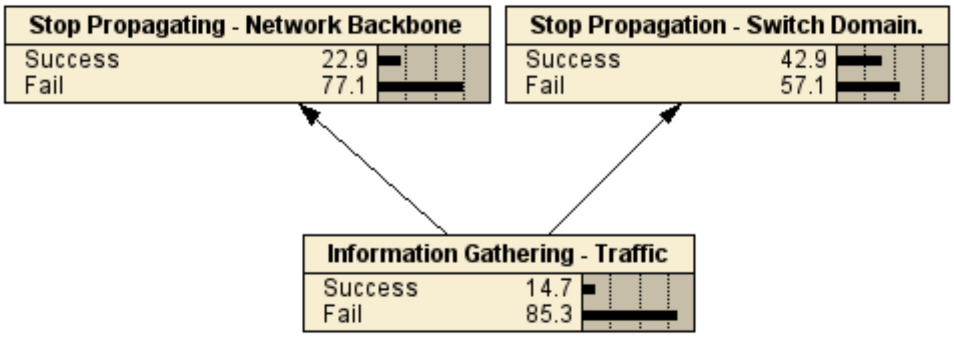
$$R = w \cdot [P(A), P(B) \ldots P(H)] \cdot \iota; \; (R, w) \in [0, 3] \tag{8.6}$$

Which will compute the risk considering all possible threats towards the enterprise network. Not all threats should be considered, few threat scenarios and the risk they pose to the mechanism can be computed the same way. For example, assuming asset weight $w = 3$ and network or mechanism parameters $\beta = 0.7$, $\gamma = 0.6$, $\epsilon = 0.6$, and $\iota = 0.9$ to calculate the risk of a malicious intruder compromising the MAC algorithm by controlling the agent itself (scenario 9, see A.1.2), while another intruder forming a DoS attack on Seawave by manipulating STP next bridge field in the switch (scenario 14, see A.1.3) and another trying to stop Seawave from propagating over the network backbone by manipulating the ARP table stored in the router (scenario 22, see A.1.5), we compute:

$$P(S_9) \cdot P(S_{14}) \cdot P(S_{22}) \cdot w \cdot \beta \cdot \gamma \cdot \epsilon \cdot \iota =$$
$$(0.032) \cdot (0.42) \cdot (0.036) \cdot (3.0) \cdot (0.7) \cdot (0.6) \cdot (0.6) \cdot (0.9) = 0.0329\%$$

Which results in a low level risk to the enterprise network.

8.10.2 Multiple Attack Scenarios



Stop Propagating - Network Backbone
Success 22.9
Fail 77.1
Stop Propagation - Switch Domain.
Success 42.9
Fail 57.1
Information Gathering - Traffic
Success 14.7
Fail 85.3

Figure 8.4: Bayesian Belief Threat Tree 8.6.1 without providing any evidence.

As we have constructed BB networks from single threat elements to form a threat scenario, in the same manner, we build a BB network that consists of threat scenario elements to form a multi-scenario threat to Seawave. We show how multiple adversaries try to put a halt to Seawave's propagation. Adversaries try to gather information about the mechanism, by sniffing traffic based on Goal 8.5.8 and as computed by Scenario 27 (See A.1.8); this scenario, however, is not limited to detecting the master node only, but also any type of useful information that adversaries may be able to sniff. Based on the information gathered, adversaries try to block the agents from propagating within their switch domain as described in scenario 15 (See A.1.3) and prevent the mechanism from moving forward over the enterprise backbone as described in scenario 21 (See A.1.5) , placing the mechanism into halt. Fig. 8.4 shows the multi-scenario BBN attack without entering any evidence, where scenario 15 and 21 (See A.1.3 and A.1.5 respectively) are conditionally independent given threat scenario 27 (See A.1.8). And the probabilities of the attack can be revisited using Bayesian theorem. For example suppose that we know that the mechanism has been blocked from traversing the backbone, that is scenario 21 (See A.1.5) was successful, but we do not know if there was any information gathering activity beforehand. Using Eq. (8.2) we compute:

$$P(S_{27}|S_{21}) = \frac{P(S_{21}|S_{27})P(S_{27})}{P(S_{21})} = \frac{0.4 \cdot 0.147}{0.229} = 0.26$$

The result indicates that based on the evidence that Seawave was blocked from traversing the enterprise network, the probability of information gathering activity held beforehand has risen from 0.147 up to 0.26. The marginal probability of S_{21} at Fig. 8.4 was driven using a BBN tool of inference Netica Toolset [76] and it can be computed the same way as we have shown previously (8.3). Assuming all events are true, the probability of adversaries succeeding in their attack to put the mechanism's propagation into halt is 3.53% as calculated by multiple threat scenario 28 (See A.2.1).

8.11 Summary

Using Bayesian Belief Networks, we proposed a threat analysis model to identify and quantify the threats that may occur against Seawave, combined with techniques to calculate the risk these threats may present. We have divided threats into different components to identify the goals attackers may try to achieve and designed threat scenarios (or bayesian networks) of several malicious penetrating attempts and also formed a multi-scenario threat BBN.

Several parameters (α, β, γ, δ, ϵ, ζ and ι) were used for attack performance aspects, most of them are specific to the component and another (ι) is general for the whole model. It turns out that using BBNs is very useful in building sequenced threat structure against a specific mechanism, where probabilistic inference of these threats can be driven.

The model allowed to examine and identify weak points in Seawave's configuration where an adversary can take advantage of. It also gave us the ability to predict the likelihood of a malicious attack taking place through these weak points; forming sound casual relationships between different model components to shape a whole body of security assessment process.

Chapter 9

Seawave – A Mathematical Propagation Model

9.1 Overview

In Chapter 4 and 5, we have tested Seawave under small to medium size enterprise networks only; that is because event-driven network simulations consumed a large amount of CPU power, memory, and time making it beyond our capacity to test Seawave under large scale enterprise networks. In this chapter, we propose and analyze an analytical propagation model of our vulnerability mitigation worm (Seawave). The model takes into consideration the topology structure of enterprise networks such as switches, LANs, and backbone. The model also addresses the defensive worm's delays due to CAM table reading (α), neighbor switch communication (β), and backbone mapping (ϵ). We also propose a bandwidth model to measure traffic generation within different stages of propagation. Different simulations of different hierarchical topologies of enterprise networks have been driven to further evaluate and observe Seawave's performance in large scale networks.

9.2 Introduction

In this chapter, we propose and analyze a propagation model of Seawave. We have proposed Seawave in previous chapters, however, although running topology sensitive mechanisms is better on an event driven simulator

– as it does not require the topology to be pre-deterministic, which is closer to real world networks, – these types of simulations are time and computing power consuming, which restricted Seawave to run on small to medium size enterprise networks. In this chapter, however, we study how the vulnerability mitigation worm propagates on large scale enterprise networks and analyze its stage by stage propagation before evaluating its overall performance. We also propose a bandwidth generation model to measure the amount of traffic generated by Seawave during its gradual propagation within a pre-deterministic network. In our study we take into consideration the delays of reading Content-Addressable Memory (CAM) table, Spanning Tree Protocol (STP), Address Resolution Protocol (ARP), and Open Shortest Path First (OSPF) topology information.

9.3 Related Work

The field of worm propagation modeling is not new; many models exist to progress the understanding of computer worms and how they behave. This understanding helps in designing future countermeasures that reduce or prevent the impact of malicious worms. Based on the Kermack-Mckendrick epidemic model, Zou *et al.* derive a Two-Factor Worm Model (TFWM) which simulates Code Red worm behavior. The two factors considered are the dynamic countermeasures applied by users and ISPs, as well as, the reduction in the worm's infection speed due to network congestion [116]. More modeling attempts of Code Red were proposed by Staniford *et al.* and Moore *et al.* [93, 64].

Castaneda *et al.* proposed an architecture to generate a benign worm, which acts as an active vaccination mechanism that transforms a malicious worm into a benign worm that propagates the same way as the original worm [18]. While, different Internet worm propagation models – as introduced by Zou *et al.* – have been analyzed under different scanning strategies, including: Idealized, Uniform, Divide-and-Conquer, local preference, and other scans [117].

The Passive Worm Propagation (PWP) model was proposed by Zhou *et al.* to study and analyze passive propagation techniques and its feasibility to represent a benign worm[115]. While, Fang *et al.* explored worm vaccination to counter malicious worms by proposing and analyzing three vaccination models: Running-Vacc, Noreboot-Vacc, and Reboot-Vacc [110]. In the same direction, Toutonji *et al.*, proposed a Passive Worm Propagation Quarantine (PWDQ) model, which describes a method to stop malicious worms by recovering infected hosts either by dynamic quarantine or passive benign worms techniques [101].

These models, however, do not take into consideration local topology preferences; few models briefly consider network devices such as the *Three Layer Worm Model* proposed by Su *et al.*, which is based on the Simple Epidemic Model (SEM) and the TFWM under NAT environment. The first, second, and third layers in the model represent hosts and routers, NAT hosts, and hosts under each NAT respectively [96]. But, in general, few propagation models consider the internal structure of the network, that is because many researchers look at the overall view of a worm propagating over the internet – crossing large number of interconnected networks. Therefore, from their prospective it becomes not necessary to go deep in the network architecture. That, however, does not apply to Seawave as it only operates under enterprise networks.

9.4 Epidemic Model Introduction

Indeed many techniques in the computer science field have been inspired by biology and computer worm modeling is not an exception. The way computer viruses and worms propagate are similar to their counterparts in biology. An epidemic model attacking different individuals simultaneously can describe malicious worms propagation behavior. This section briefly introduces three epidemic models that pave the ground for Seawave's propagation model. The terminology used in these three models is as follows:

- *I(t)*: The number of infectious nodes at time t.
- *S(t)*: The number of susceptible nodes at time t.
- *N*: Total number of targeted nodes (the size of the scope).
- η: Average scanning rate.

9.4.1 Simple Epidemic Model in a Homogeneous System

In this model nodes are of two states either infectious I(t) or susceptible [N - I(t)] and when a node is infected it remains infectious and does not get removed. As the system is homogeneous each node has equal probability to contact any other node. The simple epidemic model can be expressed as [25]:

$$\frac{dI(t)}{dt} = \eta I(t)[N - I(t)] \tag{9.1}$$

At *t=0* there are *I(0)* infectious nodes and the rest *[N - I(0)]* are susceptible.

9.4.2 Flash Worm

Perhaps one of the fastest worms – as target addresses are already known – Staniford *et al.* have introduced what they termed as *Flash Worm* [93]. The worm extends the technique of sized hit-list to cover the whole scope saving the worm the effort of vulnerability scanning and allocating its victims. Based on the simple epidemic model, Zou *et al.* derive its propagation model [117]:

$$\frac{dI(t)}{dt} = \frac{\eta}{N} I(t)[N - I(t)] \tag{9.2}$$

They showed that without considering delay and with N = 360,000, η = 358/min, and *I(0)* = 10 the flash worm can infect 99% of its targets at T = 2.23 seconds.

9.4.3 Perfect Worm

Faster than the Flash worm comes the perfect worm; proposed by Zou *et al.* as the fastest propagation worm. The perfect worm – where our propagation model is based – knows all its targets like the Flash worm, however, the infected nodes cooperate with each other such that there are no reinfections (i.e. no wasted scans). The propagation model of the worm is [117]:

$$\frac{dI(t)}{dt} = \begin{cases} \eta I(t); & I(t) < N, \\ 0; & I(t) = N \end{cases} \tag{9.3}$$

Assuming the worm starts with I(0) infected nodes, the analytical solution becomes:

$$I(t) = min[I(0)e^{\eta t}, N]. \tag{9.4}$$

9.5 Seawave Propagation Model

In this section we will propose an analytical propagation model of Seawave. Since Seawave does not infect but rather vaccinate vulnerable nodes from potential and present malicious attacks, we provide a slightly different terminology than common epidemic terminology:

- ***V(t)*** The number of vaccinated nodes at time t.
- **A(t)** The number of Agent nodes at time t.
- ***S(t)*** The number of susceptible nodes at time t.

- ***P(t)*** The number of packets generated at time t. Excluding within switch bandwidth.
- ***L(t)*** The network topology level of operation at time t – the front line of propagation.
- **Switch(L(t))** Number of switches at level $L(t)$.
- ***N*** Size of the switch (i.e. the number of nodes connected to the switch).
- ***LAN*** The list of LANs within the enterprise network.
- ***K*** Number of LANs in the enterprise network.
- η Average scanning rate within a switch.

In the model there are three states of the node:

- **Agent:** A node becomes an agent, when another agent node self-replicates to it. An agent node is responsible of further propagation to neighboring switches and vaccinating other susceptible nodes within the same switch.
- **Vaccinated:** A node that has been vaccinated by an agent node.
- **Susceptible:** A node that is vulnerable to malicious attacks.

Agent and vaccinated nodes continue to remain in their state during the propagation. Since Seawave depends on the topology of the network for its propagation we have to specify the topology characteristics; the enterprise network consists of K LANs each LAN is of the size $LAN_1, LAN_2, .., LAN_K$ respectively. We mean by LAN size, the number of interconnected switches.

Seawave propagates gradually level after level, from switch to neighboring switches and further; Fig. 9.1 illustrates what we mean by levels. We assume the following for the model:

- All nodes are susceptible.

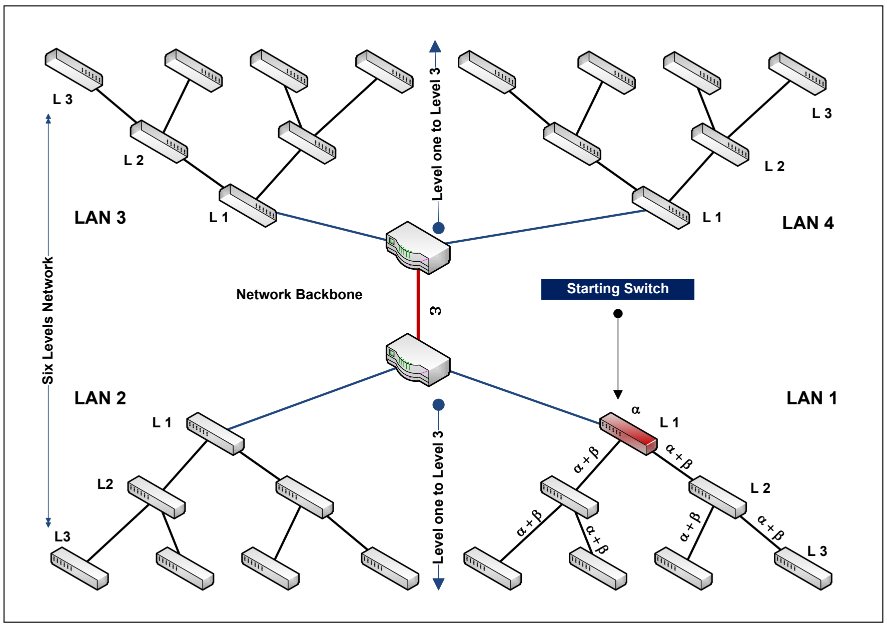


Figure 9.1: Enterprise Network Example

- The topology implements the Spanning Tree Protocol.
- All switches are of fixed size N.

We also assume that Seawave initially covers LAN_1 starting from the first level $L(t) = 1$, before propagating to other LANs within the enterprise network. Upon covering LAN_1 agents self-replicate to other LANs simultaneously and propagate until all LANs are covered.

Since the vaccination process is within the switch, this gives the agent the ability to vaccinate all nodes connected to the switch as soon as it learns their Media Access Control (MAC) addresses retrieved from the CAM table. This can be modeled by (9.3), since agents know all their targets and do not do any revaccination, see Figure 9.2. Let t_{sf} denote the time needed for the agent to vaccine all nodes connected to the switch, such that $V(t_{sf}) = N$; based on (9.4) the number of vaccinated nodes in level $L(t) = 1$ (the initial switch) is therefore:

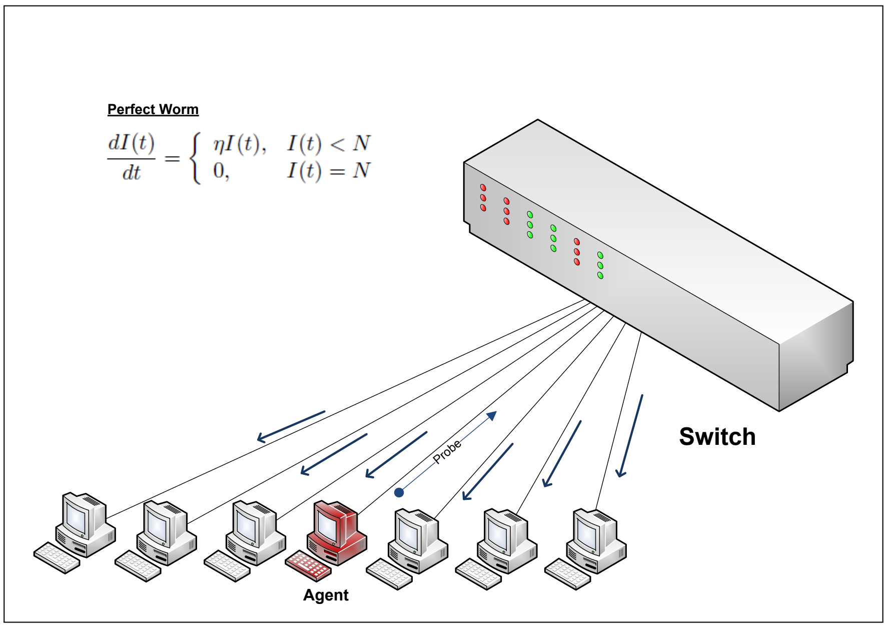

Perfect Worm
$\frac{dI(t)}{dt} = \begin{cases} \eta I(t), & I(t) < N \\ 0, & I(t) = N \end{cases}$
Probe
Switch
Agent

Figure 9.2: Within Switch Vaccination

$$V(t) = min[I(0)e^{\eta t}, N]; t \leq t_{sf} \tag{9.5}$$

Without considering network delays, the start time for Seawave to propagate to level $L(t)$ can be derived by:

$$LT(t) = t_{sf} \cdot (L(t) - 1); \quad t > t_{sf} \tag{9.6}$$

Since the mechanism covers the enterprise network level by level $L(t_0)$, $L(t_1)$, $L(t_2) \ldots L(t_{nf})$ we derive $L(t)$:

$$L(t) = \lceil t/(t_{sf} + \alpha + \beta) \rceil \tag{9.7}$$

Where α and β are delay parameters described later in this section. Since all agents start vaccination on the same level $L(t)$ in parallel, then based on (9.6) and (9.7) the number of vaccinated nodes on $L(t)$ only, can be obtained by:

$$Switch(L(t)) \cdot V(t - LT(t)) \tag{9.8}$$

Let t_{nf} denotes the total time to cover the whole enterprise network the total number of vaccinated nodes up to $L(t) - 1$ becomes:

$$N \times \sum_{i=L(t_0)}^{L(t)-1} Switch(i);\ t \leq t_{nf} \tag{9.9}$$

Adding both (9.8) and (9.9) gives us the number of vaccinated nodes beyond the first level $L(t) > 1$:

$$Switch(L(t)) \cdot V(t - LT(t)) + N \times \sum_{i=L(t_0)}^{L(t)-1} Switch(i);\ t_{st} < t \leq t_{nf} \tag{9.10}$$

And when $t > t_{nf}$, then all enterprise network nodes have been vaccinated:

$$N \times \sum_{i=L(t_0)}^{L(t_{nf})} Switch(i);\ t > t_{nf} \tag{9.11}$$

Therefore, by combining (9.5), (9.10), and (9.11) we derive the propagation model of Seawave:

$$V(t) = \begin{cases} min[I(0)e^{\eta t}, N]; & t \leq t_{sf} \\ Switch(L(t)) \cdot min[I(0)e^{\eta(t-LT(t))}, N] & \\ \qquad + N \times \sum_{i=L(t_0)}^{L(t)-1} Switch(i); & \\ & t_{st} < t \leq t_{nf} \\ N \times \sum_{i=L(t_0)}^{L(t_{nf})} Switch(i); & t > t_{nf} \end{cases} \tag{9.12}$$

We also can compute the number of agent nodes scattered around the enterprise network by counting the number of switches at t:

$$A(t) = \sum_{i=L(t_0)}^{L(t)} Switch(i);\ \ t \leq t_{nf} \tag{9.13}$$

And the number of susceptible nodes at time t, can be derived by subtracting the number of vaccinated nodes from the total number of nodes in the whole network:

$$S(t) = (N \times \sum_{i=L(t_0)}^{L(t_{nf})} Switch(i)) - V(t) \tag{9.14}$$

However, the propagation model (9.12) does not consider the following time delays during propagation:

1. Time for the agent to fetch the CAM table from the directly connected switch (α).
2. Time for the agent to fetch CAM tables and STP information from neighboring switches and self-replicating to nodes in neighboring switches (β).
3. Time for the agent to fetch ARP and OSPF information from routers in the backbone and self-replicating to other LANs in the enterprise network (ϵ).

Since these delays occur on different stages of the propagation, then the delay within the start switch at $L(t) = 1$ is:

$$t - \alpha; \quad t \leq t_{sf} \; where \; V(t - \alpha) = 0, \forall t < \alpha \tag{9.15}$$

Assuming that the time to cover LAN_1 is t_{LAN1}, the time during different propagation levels within LAN_1 becomes:

$$t - [L(t) \cdot \alpha] - [(L(t) - 1) \cdot \beta]; \quad t_{sf} < t \leq t_{LAN1} \tag{9.16}$$

And up to t_{nf} the propagation time after delays becomes:

$$t - [L(t-\epsilon)\cdot\alpha] - [(L(t-\epsilon)-2)\cdot\beta] - \epsilon; \quad t_{LAN1} < t \le t_{nf} \tag{9.17}$$

Based on (9.15), (9.16), and (9.17) we can derive t on different propagation levels as follows:

$$t = \begin{cases} t-\alpha; \\ \qquad t \le t_{sf} \; where \; V(t-\alpha) = 0; \forall t < \alpha, \\ t - [L(t)\cdot\alpha] - [(L(t)-1)\cdot\beta]; \\ \qquad\qquad t_{sf} < t \le t_{LAN1}, \\ t - [L(t-\epsilon)\cdot\alpha] - [(L(t-\epsilon)-2)\cdot\beta] - \epsilon; \\ \qquad\qquad\qquad t_{LAN1} < t \le t_{nf}. \end{cases} \tag{9.18}$$

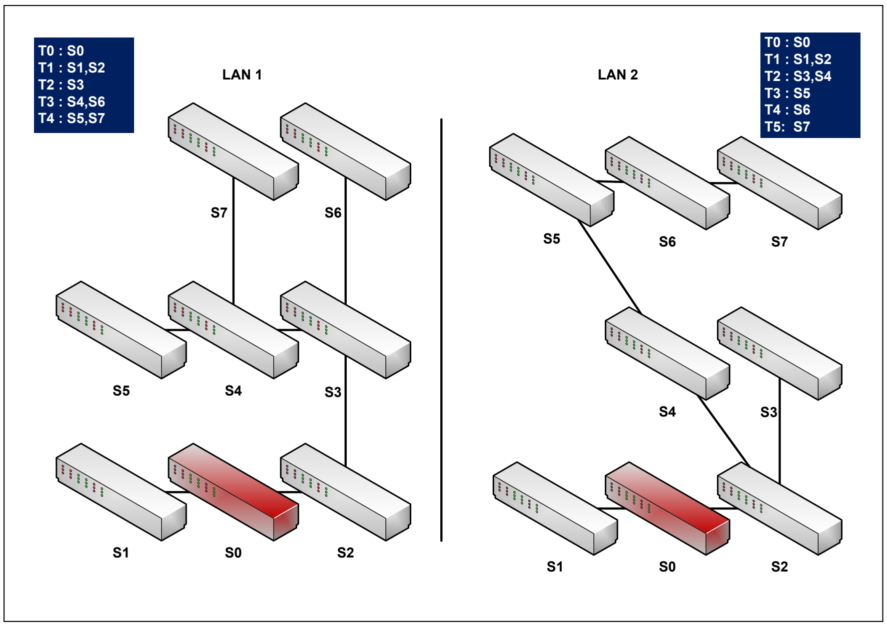


Figure 9.3: Two LANs of the same number of switches and links but different time coverage.

9.5.1 Network Topology Model

Topology dependent self-replicating self-propagating mechanisms are sensitive to network topologies (see Chapter 6, Section 6.8.1), as they define their propagation path based on topology information they self-discover during their gradual dissemination. As it shows in Fig. 9.3, Seawave will require more time to cover LAN_2 compared to LAN_1, although both layouts have the same number of switches and links. Therefore, it is necessary to define the enterprise network in detailed manner for the propagation model to utilize. However, it is not feasible to construct non-deterministic topologies, as due to their unpredictable nature they would require large quantities of topology descriptions. Therefore, it is more feasible to construct topologies of a specific layout for the propagation model to walk through and to ease the task of resolving $Switch(L(t))$. Below are characteristics of topologies we have used:

1. Each switch has two neighbors, except edge switches.
2. Each router in the backbone connects at most two LANs.

Fig. 9.1 shows an example of a topology of such characteristics. Based on these attributes, we can calculate the total number of levels Seawave go through to cover the whole enterprise network:

$$\lceil \log(LAN_1) \rceil + \lceil \log(max[LAN - \{LAN_1\}]) \rceil \tag{9.19}$$

While, the number of switches just before the last level of LAN_1 can be computed by:

$$2^{L(t)-1}; \quad L(t) < \lceil \log(LAN_1) \rceil \tag{9.20}$$

Further, the number of switches of the last propagation level of LAN_1 (edge level) becomes:

$$LAN_1 - (2^{\lfloor \log(LAN_1) \rfloor} - 1); \quad L(t) = \lceil \log(LAN_1) \rceil \tag{9.21}$$

One level further leaves us with switches equal to the number of LANs else LAN_1:

$$K-1; \quad L(t) - \lceil \log(LAN_1) \rceil = 1 \tag{9.22}$$

When Seawave bypasses the backbone and passes the first level after LAN_1 but does not reach an edge level of LAN_i, the number of switches becomes:

$$\sum_{i=2}^{K} 2^{(L(t) - \lceil \log(LAN_1) \rceil - 1)}; \quad L(t) - \lceil \log(LAN_1) \rceil < \lceil \log(LAN_i) \rceil \tag{9.23}$$

Where at the end edge of LAN_i, the number of switches can be computed by:

$$\sum_{i=2}^{K} LAN_i - (2^{\lfloor \log(LAN_i) \rfloor} - 1); \quad L(t) - \lceil \log(LAN_1) \rceil = \lceil \log(LAN_i) \rceil \tag{9.24}$$

Of course, number of switches becomes 0 when $L(t)$ is beyond the scope of Seawave:

$$0; \quad L(t) - \lceil \log(LAN_1) \rceil > \lceil \log(LAN_i) \rceil \tag{9.25}$$

Based on (9.20), (9.21), (9.22), (9.23), (9.24), and (9.25) we can compute the number of switches at different propagation levels:

(1) Seawave's propagating upto $(L(t_{LAN_1})+1)$: $When\ t \le t_{LAN1} + \epsilon + (t_{sf} + \alpha + \beta)$

$$Switch(t) = \begin{cases} 2^{L(t)-1}; L(t) < \lceil \log(LAN_1) \rceil, \\ LAN_1 - (2^{\lfloor \log(LAN_1) \rfloor} - 1); L(t) = \lceil \log(LAN_1) \rceil, \\ K-1; L(t) - \lceil \log(LAN_1) \rceil = 1. \end{cases} \tag{9.26}$$

(2) Seawave's propagating from $(L(t_{LAN_1}) + 1)$ upto $L(t_{LAN_{nf}})$:
$When\ t > t_{LAN1} + \epsilon + (t_{sf} + \alpha + \beta)$

$Switch(t) =$

$$\sum_{i=2}^{K} \begin{cases} 2^{(L(t)-\lceil\log(LAN_1)\rceil-1)}; L(t) - \lceil\log(LAN_1)\rceil < \lceil\log(LAN_i)\rceil, \\ LAN_i - (2^{\lfloor\log(LAN_i)\rfloor} - 1); L(t) - \lceil\log(LAN_1)\rceil = \lceil\log(LAN_i)\rceil, \\ 0; L(t) - \lceil\log(LAN_1)\rceil > \lceil\log(LAN_i)\rceil. \end{cases} \tag{9.27}$$

Now as we have defined $Switch(t)$, we are ready to run our simulations.

9.5.2 Seawave Bandwidth Model

Following bandwidth and communication assumptions in Chapter 5, if we denote R as the number of routers in the backbone, then the total number of packets generated by Seawave to cover the whole enterprise network can be derived by:

$$P(t_{nf}) = 4 \times [(\sum_{i=1}^{K} LAN_i) - K] + 2 \times R + 2 \times (K-1) \tag{9.28}$$

However, to derive the bandwidth based on time t, we have to take into consideration the levels of propagation Seawave goes through. Within LAN_1 the amount of bandwidth can be computed by:

$$(2^{L(t)} - 2) \times 4; \quad t_{sf} < t \leq t_{LAN1} \tag{9.29}$$

And beyond LAN_1 the bandwidth becomes:

$$4 \times [(\sum_{i=L(t_0)}^{L(t)} Switch(i)) - K] + 2 \times R + 2 \times (K-1); t_{LAN1} < t \leq t_{nf} \tag{9.30}$$

We ignore packets generated within the switch as they do not have a significant impact on network main links; therefore, packets within the first switch are 0. Based on (9.29) and (9.30) we derive the bandwidth model:

$$P(t) = \begin{cases} 0; \quad t \leq t_{sf}, \\ (2^{L(t)} - 2) \times 4; \quad t_{sf} < t \leq t_{LAN1}, \\ 4 \times [(\sum_{i=L(t_0)}^{L(t)} Switch(i)) - K] + 2 \times R \\ \qquad +2 \times (K-1); \quad t_{LAN1} < t \leq t_{nf}. \end{cases} \tag{9.31}$$

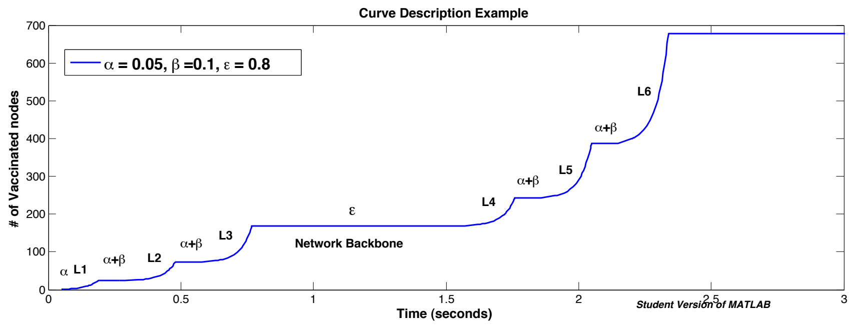
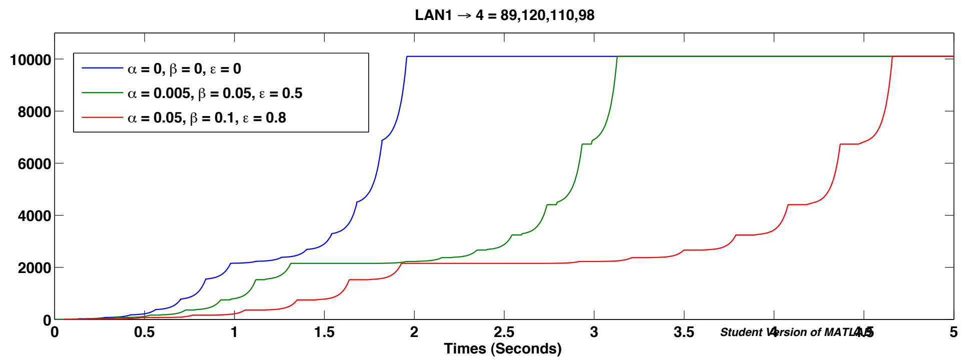


Figure 9.4: Curve Description of a small six level network as in Fig. 9.1

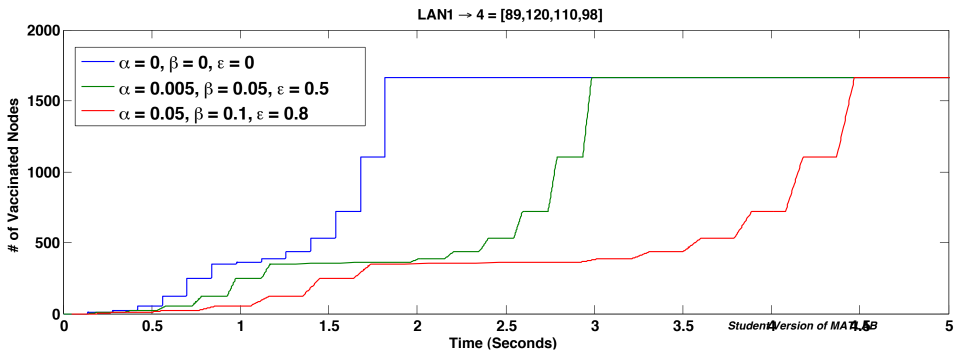
9.6 Simulation Results

After defining the propagation and the bandwidth models, we are ready to run our simulations using Matlab [59]. It is, however, useful to describe Seawave's curve for more clarity. Fig. 9.4 shows a simple curve of the network example provided at Fig. 9.1, we have plotted the delays for better readability. Notice the waves in the curve are the result of level by level propagation. At the very start of the curve only α is applied, as Seawave still operate within the first switch at $L(t) = 1$; after which it moves to the next level ($L(t) = 2$), where $\alpha + \beta$ are applied to include neighbor switch communication, and so on. And when Seawave reaches the backbone, the delay ϵ is applied before propagating to all other LANs within the enterprise network at $L(t) = 4$ and it continues until the whole network is covered at $L(t) = 6$.

Assuming $\eta = 23$ and $I(0) = 1$, we run Seawave on different network topologies, with different number of switches, LANs, and delays. We also



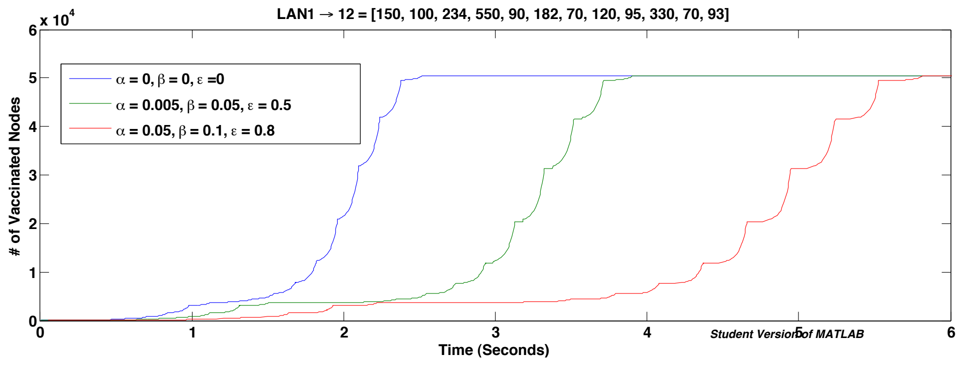
(a) Seawave Propagation



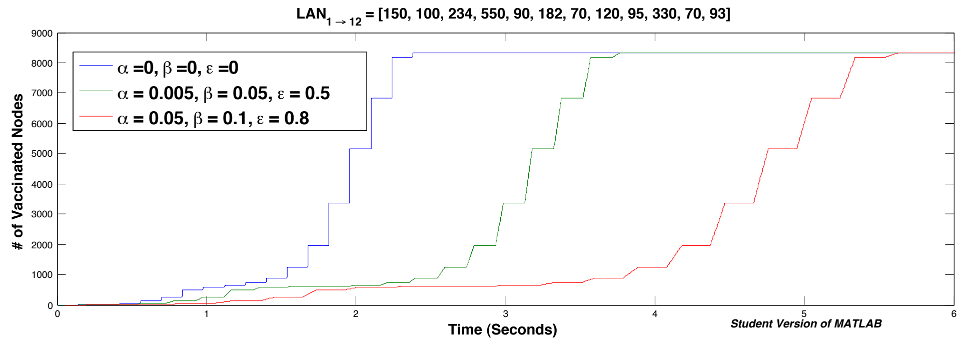
(b) Seawave Bandwidth

Figure 9.5: Network of 10,000 nodes distributed among 4 LANs with inside switch, switch to switch, and backbone delays.

show the amount of bandwidth generated on these topologies. In a network of 10,000 vulnerable nodes distributed among 4 LANs and 417 switches, the time required to cover the whole network was 2, 3.2, and 4.7 seconds based on $\alpha = 0$ $\beta = 0$ $\epsilon = 0$, $\alpha = 0.005$ $\beta = 0.05$ $\epsilon = 0.5$, and $\alpha = 0.05$ $\beta = 0.1$ $\epsilon = 0.8$, respectively; and the bandwidth generated was 1662 packets, see Fig. 9.5a and 9.5b for more details. In a network of 2084 switches (50,000 nodes) distributed among 12 LANs, based on the previous delays the time it took to cover the network was 2.5, 3.96, and 5.87 seconds, respectively, with a bandwidth of 8322 packets; more details can be found in Fig. 9.6. Another network of 21 LANs and 6250 switches (150,000 nodes) has required 2.86, 4.36, and 6.46 seconds on the same delays. The same network has required 24978 packets to become covered, more results are given in Fig. 9.7.



(a) Seawave Propagation



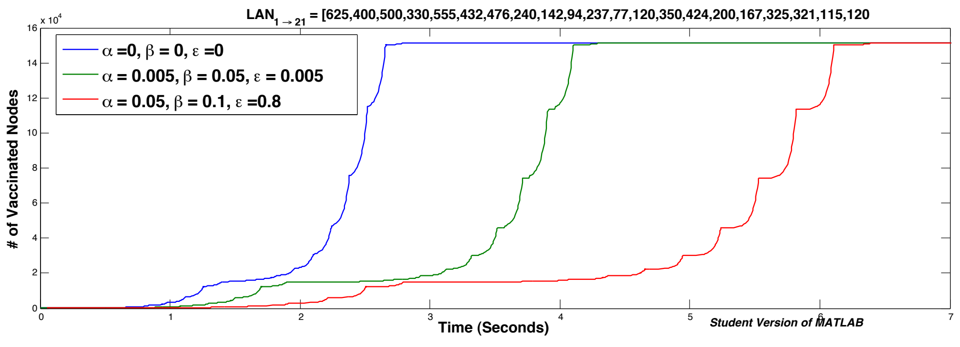
(b) Seawave Bandwidth

Figure 9.6: Network of 50,000 nodes distributed among 12 LANs with inside switch, switch to switch, and backbone delays.

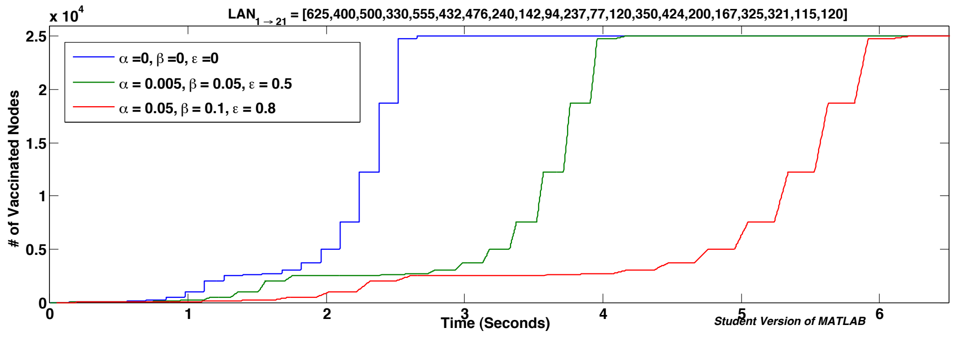
The time observed to cover a network of 500,000 nodes and 20834 switches distributed among 70 LANs with different delays was 2.86, 4.36, and 6.46 seconds respectively. The same network consisted of 20 levels and generated 83264 packets bandwidth, further details can be viewed at Fig. 9.8. Then we finally run Seawave on a one million node network connected to 41668 Switches and 140 LANs forming 18 levels of propagation; the time observed to cover the whole enterprise network was 2.52, 3.91, and 5.82 seconds respectively. And it took 166530 packets to cover the network.

9.7 Discussion

Indeed network topologies and structures affect the way how Seawave propagates, as its self-discovery nature binds it to the topology information grad-



(a) Seawave Propagation

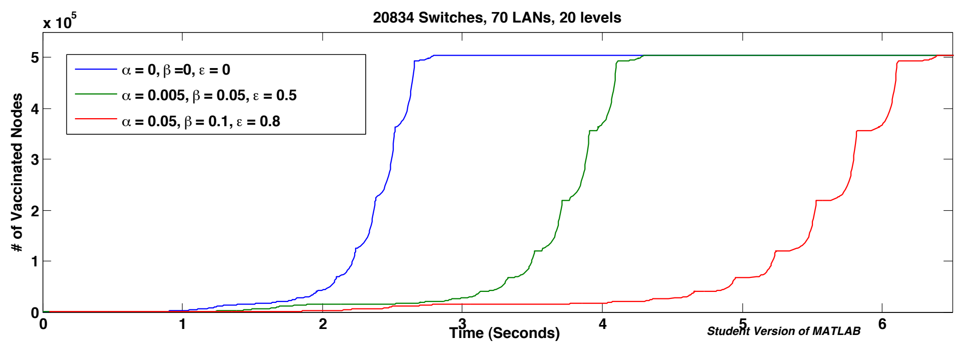


(b) Seawave Bandwidth

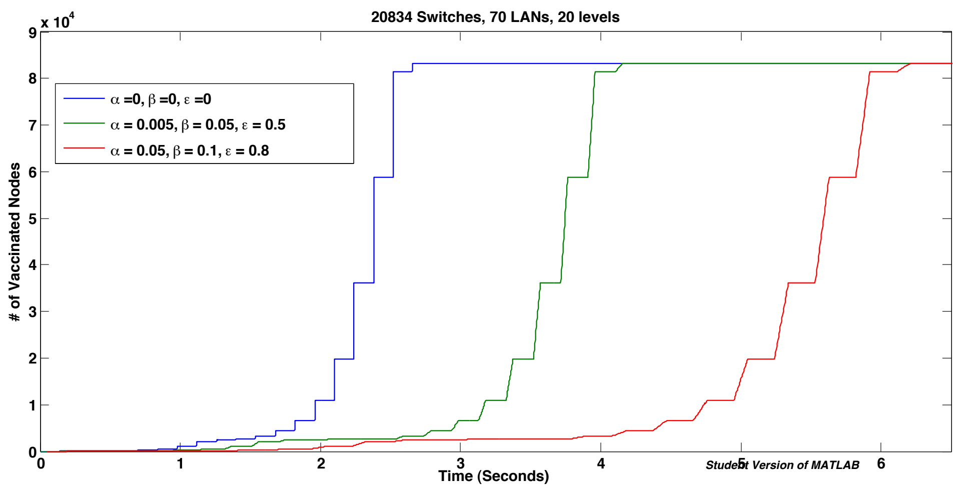
Figure 9.7: Network of 150,000 nodes distributed among 21 LANs with inside switch, switch to switch, and backbone delays.

ually revealed during its propagation. That is why it is more accurate to test such mechanisms with an event driven simulator (such as NS2 [42]) as you can run these types of systems on different non-deterministic topologies – which have been done in Chapter 4 and 5. However, these types of simulators consume high amount of memory and CPU power which makes running Seawave on large enterprise networks very time consuming. Therefore, an analytical propagation model is needed to measure Seawave's performance on very large networks.

Running the mechanism on a network of 7 levels and 500 nodes (21 Switches), yields 1.76 seconds time coverage under parameters $\alpha = 0.005$, $\beta = 0.05$, $\epsilon = 0.5$, which is very close to the time coverage observed using a discrete-event simulator, which was 1.86 seconds on a 500 node network (see Chapter 7). During simulations we have observed that the po-



(a) Seawave Propagation

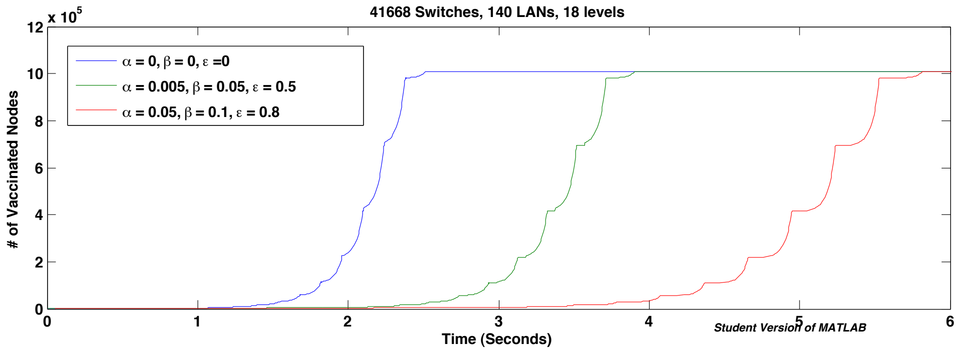


(b) Seawave Bandwidth

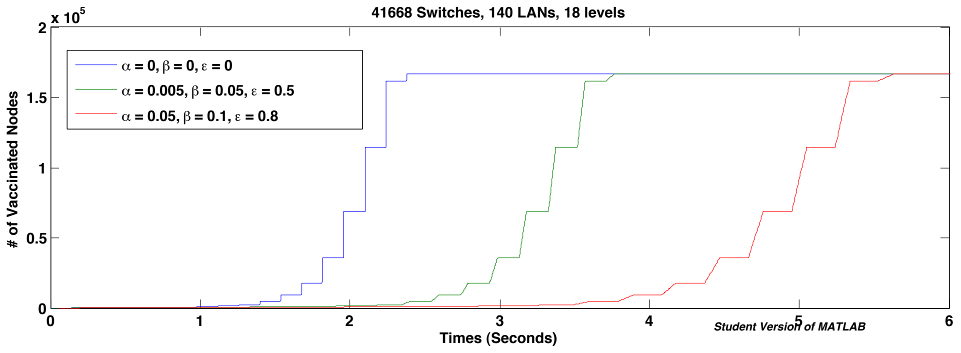
Figure 9.8: Network of 500,000 nodes distributed among 70 LANs with inside switch, switch to switch, and backbone delays.

sition where the first agent is located does affect the propagation speed, as the closer the agent is to the backbone router the sooner Seawave spreads around the enterprise network. That is because the mechanism becomes able to draw the network map based on the information retrieved from the backbone routers in the early stages of propagation. Therefore, for faster propagation the security team might consider installing the starting agent on the switch directly connected to the router, or as a possible future improvement of Seawave, the agent could try to detect the default router and start mapping the backbone as early as possible.

It has been observed that the smaller LAN_1 is the faster the propagation



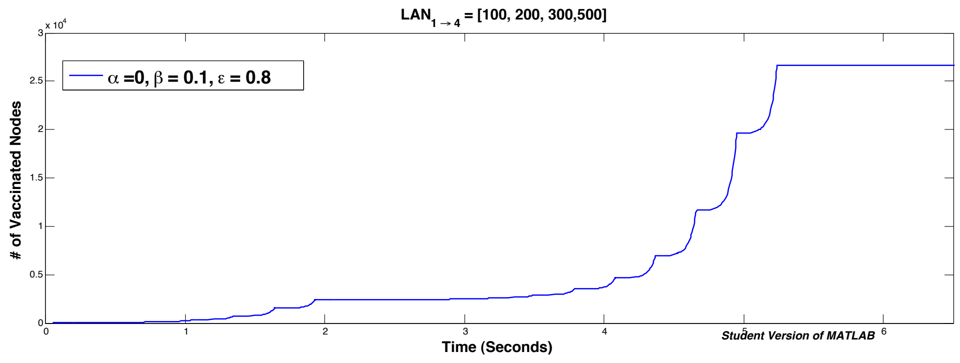
(a) Seawave Propagation



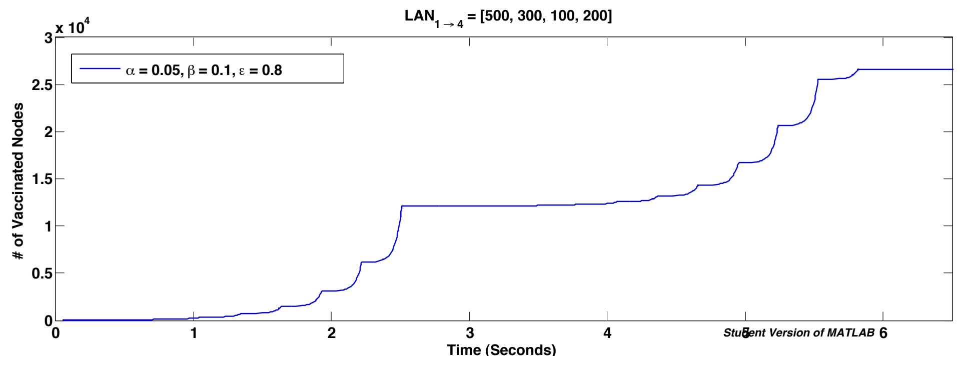
(b) Seawave Bandwidth

Figure 9.9: Network of 1000,000 nodes distributed among 140 LANs with inside switch, switch to switch, and backbone delays.

– for the same previous reasons (i.e. closer to the backbone). Not smaller in the number of nodes but in propagation levels; as Seawave is affected more by the levels of propagation rather than number of nodes, that is due to the nature of self-replicating to all switches on the same level leading the vaccination process to becoming simultaneous. We can see this clearly in Fig. 9.8 (500,000) and Fig. 9.9 (one million) which resulted in 4.36 and 3.91 seconds time coverage respectively – based on the exact parameters. Although both networks maintain a 50% number of nodes difference, the 500,000 nodes network had 20 levels compared with the one million node network which only had 18 levels from the prospective of Seawave. Even if the number of levels is the same, the position where Seawave starts propagating impacts its general behavior, producing different performance results depending on



(a) $LAN_{1\to4}$ = 100, 200, 300, 500



(b) $LAN_{1\to4}$ = 500, 300, 100, 200

Figure 9.10: Two different curves of the same network, but with different Seawave start points.

the start point of propagation. We can see this in Fig. 9.10a and 9.10b which shows different curves of the same network, however, with different starting points ($LAN_1 = 100$ and $LAN_1 = 500$). And it shows different time coverage, due to different propagation levels, as based on (9.19) it took Seawave 16 levels to cover the network starting at $LAN_1 = 100$ as in Fig. 9.10a and 18 levels to cover the same network starting at $LAN_1 = 500$ as in Fig. 9.10b.

9.8 Summary

In this chapter, we have proposed a propagation model of our vulnerability mitigation worm named Seawave. In the model we have considered the small details of a network structure such as the number of switches, LANs, and backbone. Also we have considered the delays to vaccine switch nodes (α), self-replicate to neighboring switches (β), and bypass the network backbone (ϵ). We have also proposed a bandwidth model to measure the amount of traffic generated, in addition to, defining a network topology model to run under.

We have run Seawave on networks of different sizes starting from 10,000 to one million nodes and we have spotted some potential improvements to the mechanism, such as adding the agent capability to probe for the backbone router during early stages of propagation. And we have observed how the starting position affects the performance and how network levels influence the time coverage more than node numbers.

Chapter 10

Summary and Conclusions

10.1 Summary

In this thesis we have revised our prejudgments towards computer worms and tried to reassess their capabilities, however not as a malware, but as a vulnerability mitigation mechanism. Chapter 2, highlighted the need to revisit the definitions of self-replicating programs, e.g. viruses, worms, and botnets, and extend them to include the defensive prospective. We have also proposed general design guidelines for defensive worms, before briefly addressing the fear factor that usually prevents the industry from adopting such defensive approaches. Chapter 3 highlighted different attempts that considered defensive worms for vulnerability mitigation, countermeasures, or administrative tasks by pointing out researchers work in that regard.

After giving the reader an overview of defensive worms and previous research on that topic, we proposed a novel controlled self-replicating, self-propagating, self-contained vulnerability mitigation mechanism (i.e. defensive worm) named *Seawave* in Chapter 4. Seawave utilizes CAM, STP, OSPF, and ARP protocols to traverse the enterprise network. Further improvements to our vulnerability mitigation worm have also been addressed in Chapter 5, before briefly proposing another novel defensive worm that utilize information retrieved by the LLDP protocol to propagate in Chapter 6.

In Chapter 7, thereafter, we have released Seawave against a malicious random scanning worm (that mimics to some extent Slammer worm behavior) and evaluated its performance and observed its capabilities in defending the enterprise network against such malicious outbreak. Where in Chapter 8 we proposed a Bayseian Belief Networks based threat model to address different probabilistic scenarios to breach Seawave. The chapter provides sequential attacks in the form of baysian trees and measure the likelihood of these – scenario based – attacks taking place. The model also addresses the possibility of using different small scale attack scenarios as part of a major operation – launched by multiple adversaries – to compromise Seawave.

Finally in Chapter 9 we proposed an analytical propagation model to evaluate and analyze Seawave's performance on large scale enterprise networks. The mechanism has been examined under different sizes of networks, starting form 10,000 nodes up to one million. The chapter also proposed a bandwidth generation model to measure traffic generation within different stages of Seawave propagation – excluding packets generated within the switch.

10.2 Directions for future work

The topic of designing computer worms for defensive use can not be described as mature, further work needs to be accomplished by the research community to be able to tackle all the problems associated with self-replicating and self-propagating network programs and pave the way for the industry to adopt such approaches. Our future work shall be in that direction, specifically:

10.2.1 Seawave in Industrial Standards.

Although we have run Seawave on different event-driven environments that mimics to a large extent real world enterprise networks, deploying the vulnerability mitigation mechanism in an actual real world network is necessary to set it up for industry standards and make sure it operates within acceptable protective measures. Introducing the first reliable self-replicating and self-propagating security product is important to move this topic from research to real world networks. Further enhancements to the defensive worm shall be introduced to make it more robust and more capable in working under different environments.

10.2.2 Further Work on LLDP Vulnerability Mitigation Defensive Worm.

In the thesis we have introduced a vulnerability mitigation mechanism that utilized LLDP for its propagation and we believe there is more space for improvements, including:

- Reduce the number of redundant probes.
- Add the capability for the mechanism to traverse the backbone and cover the whole enterprise network.
- Assess the threats towards the defensive worm.
- Provide a mathematical propagation model for evaluation in large scale networks.

10.2.3 Wireless Defensive Worms.

We have only considered defensive worms for wired networks, however, it is necessary to address the wireless medium as well. One possible approach would be designing an LLDP-Based defensive worm to mitigate vulnerabilities in LLDP-Supported Wireless networks, where the worm propagate depending on information retrieved from the LLDP protocol.

10.3 Conclusion

The race between malicious intrusions and defensive mechanisms is always close. However, threats are increasing in scale, intelligence, and sophistication and unless the research community push forward new defensive ideas and techniques, the intruders will always be in the lead. Several calls have been issued by industries and governments for solutions that provide prompt and effective responses to prevent (or even reduce) the damage caused by continuous malicious attacks. Unfortunately, researchers seldom look into worms beyond the malicious prospective; indeed that limited view has led the research community to overlook the possible employment of their distinctive features in network protection.

In this thesis we have revisited the definitions of worms and self replicating code – in general –, to pave the way for a straightforward identification of viruses, worms, and botnets; and to also consider the beneficial sides of these programs, which is often overlooked in the literature. The proposed definitions and taxonomy of self-replicating programs are based on their properties, while the distinction between malicious and defensive (or beneficial) actions is left for the jurisdiction. This is necessary as in this topic the ethical side is always highlighted, although the common prospective in regard to self-replicating code is often negative – disallowing their commercial use. We have also observed through the related work chapter, that the concept of using defensive worms as a countermeasure is not new but indeed not mature enough.

We have looked into worms free from any prejudgments and tried to hire their capabilities to mitigate vulnerabilities in enterprise networks, by proposing a controlled self-replicating, self-propagating, and self-contained vulnerability mitigation worm by the name of Seawave. To the best of our knowledge the mechanism is the first worm that utilize the second layer of the OSI model (Data link layer) as its main propagation medium, thats because it is meant to be deployed within enterprise networks and not on the Internet. We found that to tackle the problem of excessive bandwidth generation – usually associated with worms – the propagation should not be viral and should adhere to control measures, such as using topology information as in the case of Seawave. The worm also can be monitored and supervised by allowing agents to refer to a master entity for instructions.

Owing to the self-replicating and self-propagating properties of Seawave, the mechanism possess the capabilities of:

- **Short distance communication with vulnerable nodes.**
- **Intermittent nodes vulnerability detection.**
- **Network topology discovery.**
- **Intelligent Network Propagation.**
- **Workload distribution.**

It worth mentioning that the network topology discovery feature would aid in solving the growing problem of allocating and disinfecting malicious botnets scattered around networks.

By modeling several statistical attack scenarios against our vulnerability mitigation mechanism, weak and strong points becomes more apparent. It is observed that most weaknesses do not come from Seawave itself, but from the topology it runs within; thats because topology dependent self-replicating and self-propagating mechanisms becomes vulnerable to the same vulnerabilities of the topologies they run within and any protective measures applied to these topologies, eventually becomes protective measures to these mechanisms.

For topology sensitive mechanisms, such as Seawave, it is better to asses the performance within an event-driven simulation environment using different – randomly generated – topologies. As in event-driven simulations the topology is not required to be pre-deterministic, which is closer to real world networks; and that what we have done for small and medium scale enterprise networks. However due to lack of resources, doing the same assessment on large scale networks becomes difficult. Therefore, we have run Seawave on large scale enterprise networks by proposing a mathematical propagation model where we have observed different points of improvements. In the model Seawave run within a pre-deterministic topology with different sizes up to one million nodes. The propagation model takes into consideration the internal details of the topology, since our defensive worm is topology sensitive; and since Seawave follow the STP in its propagation, it views the enterprise network in terms of levels and not necessary number of nodes. The more levels the network has, the more time it takes the mechanism to cover the network. Furthermore, the number, structure, and size of LANs within the enterprise network, in addition to, the position where the mechanism is initiated does also affects the performance of the defensive worm, including its time of coverage.

In general, we believe that self-replicating code for defensive (or beneficial) purposes worth considering to become a new research direction in academia. Seawave has been designed to assess enterprise networks security teams in their work to mitigate vulnerabilities. Several simulations and evaluations of the defensive worm have shown promising results in providing rapid and effective vulnerability mitigation coverage, in addition to, controlled propagation. Which encourages exploring this field of study and open new paths to solve modern network problems; as the common way of thinking has created the problem and unless it changes, the problem might remain.

Appendix A

Further Results

A.1 Attack Scenarios Probabilities

A.1.1 Goal: Malicious Use of Agent to Master Node Communication

Scenario 1 Probability of decrypting the Agent-Master node communication line by breaking asymmetric RSA encryption mathematically.

$$\begin{aligned}
&= P(A \cap B \cap \neg C \cap D) \\
&= P(A|B \cap \neg C \cap D) \cdot P(B \cap \neg C \cap D) \\
&= P(A|B) \cdot P(B \cap \neg C \cap D) \\
&= P(A|B) \cdot P(B|\neg C \cap D) \cdot P(\neg C \cap D) \\
&= P(A|B) \cdot P(B|\neg C \cap D) \cdot P(\neg C) \cdot P(D) \\
&= 0.1 \cdot 0.01 \cdot 0.7 \cdot 0.001 \\
&= 7 \times 10^{-5}\%
\end{aligned}$$

Scenario 2 Probability of decrypting the agent to master node communication line by breaking asymmetric encryption using brute force attack.

$$\begin{aligned}
&= P(A \cap B \cap C \cap \neg D) \\
&= P(A|B \cap C \cap \neg D) \cdot P(B \cap C \cap \neg D) \\
&= P(A|B) \cdot P(B \cap C \cap \neg D) \\
&= P(A|B) \cdot P(B|C \cap \neg D) \cdot P(C \cap \neg D) \\
&= P(A|B) \cdot P(B|C \cap \neg D) \cdot P(C) \cdot P(\neg D) \\
&= 0.1 \cdot 0.2 \cdot 0.3 \cdot 0.999 \\
&= 0.5994\%
\end{aligned}$$

Scenario 3 Probability of exhausting master node with agent connections by sending `Seawave_Agent_to_Master` packets obtained by sniffing traffic after plugging the intruder machine to the enterprise network.

$$
\begin{aligned}
&= P(G \cap H \cap I \cap \neg J \cap N \cap \neg K)\\
&= P(G|H \cap I \cap \neg J \cap N \cap \neg K) \cdot P(H \cap I \cap \neg J \cap N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H \cap I \cap \neg J \cap N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I \cap \neg J \cap N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I|\neg J \cap N \cap \neg K) \cdot P(\neg J \cap N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I|\neg J \cap N) \cdot P(\neg J) \cdot P(N) \cdot P(\neg K)\\
&= 0.5 \cdot 0.4 \cdot 0.5 \cdot 0.7 \cdot 0.5 \cdot 0.3\\
&= 1.05\%
\end{aligned}
$$

Scenario 4 Probability of exhausting master node with connections based on reply-attacks sent from a compromised node.

$$
\begin{aligned}
&= P(G \cap H \cap I \cap J \cap \neg N \cap \neg K)\\
&= P(G|H \cap I \cap J \cap \neg N \cap \neg K) \cdot P(H \cap I \cap J \cap \neg N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H \cap I \cap J \cap \neg N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I \cap J \cap \neg N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I|J \cap \neg N \cap \neg K) \cdot P(J \cap \neg N \cap \neg K)\\
&= P(G|H \cap \neg K) \cdot P(H|I) \cdot P(I|J \cap \neg N) \cdot P(J) \cdot P(\neg N) \cdot P(\neg K)\\
&= 0.5 \cdot 0.4 \cdot 0.3 \cdot 0.3 \cdot 0.5 \cdot 0.3\\
&= 0.27\%
\end{aligned}
$$

Scenario 5 Probability of exhausting master node with SYN-Flood connections.

$= P(G \cap \neg H \cap \neg I \cap \neg J \neg N \cap K)$

$= P(G|\neg H \cap \neg I \cap \neg J \cap \neg N \cap K) \cdot P(\neg H \cap \neg I \cap \neg J \cap \neg N \cap K)$

$= P(G|\neg H \cap K) \cdot P(\neg H \cap \neg I \cap \neg J \cap \neg N \cap K)$

$= P(G|\neg H \cap K) \cdot P(\neg H|\neg I \cap \neg J \cap \neg N \cap K) \cdot P(\neg I \cap \neg J \cap \neg N \cap K)$

$= P(G|\neg H \cap K) \cdot P(\neg H|\neg I) \cdot P(\neg I \cap \neg J \cap \neg N \cap K)$

$= P(G|\neg H \cap K) \cdot P(\neg H|\neg I) \cdot P(\neg I|\neg J \cap \neg N) \cdot P(\neg J \cap \neg N \cap K)$

$= P(G|\neg H \cap K) \cdot P(\neg H|\neg I) \cdot P(\neg I|\neg J \cap \neg N) \cdot P(\neg J) \cdot P(\neg N) \cdot P(K)$

$= 0.8 \cdot 0.5 \cdot 0.9 \cdot 0.7 \cdot 0.5 \cdot 0.7$

$= 8.82\%$

Scenario 6 Probability of feeding malformed information to master node.

$= P(L \cap M)$

$= P(L|M) \cdot P(M)$

$= 0.7 \cdot 0.2$

$= 14\%$

A.1.2 Goal: Malicious Use of Agent to Agent Communication

Scenario 7 Probability of stopping the agent propagation by sending the agent a `Seawave_SelfReplicate_ACK` packet by flooding the network after compromising the mechanism's mac algorithm.

$$
\begin{aligned}
&= P(A \cap B \cap C \cap D \cap \neg E \cap \neg F) \\
&= P(A|B \cap C \cap D \cap \neg E \cap \neg F) \cdot P(B \cap C \cap D \cap \neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B \cap C \cap D \cap \neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B|C \cap D \cap \neg E \cap \neg F) \cdot P(C \cap D \cap \neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B|C) \cdot P(C \cap D \cap \neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B|C) \cdot P(C|D \cap \neg E \cap \neg F) \cdot P(D \cap \neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B|C) \cdot P(C|D) \cdot P(D|\neg E \cap \neg F) \cdot P(\neg E \cap \neg F) \\
&= P(A|B \cap \neg E) \cdot P(B|C) \cdot P(C|D) \cdot P(D) \cdot P(\neg E|\neg F) \cdot P(\neg F) \\
&= 0.7 \cdot 0.5 \cdot 0.3 \cdot 0.1 \cdot 0.995 \cdot 0.8 \\
&= 0.8358\%
\end{aligned}
$$

Scenario 8 Probability of stopping the agent propagation by sending an authentic `Seawave_SelfReplicate_ACK` packet after compromising the agent.

$$
\begin{aligned}
&= P(A \cap \neg B \cap \neg C \cap \neg D \cap E \cap F) \\
&= P(A|\neg B \cap \neg C \cap \neg D \cap E \cap F) \cdot P(\neg B \cap \neg C \cap \neg D \cap E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B \cap \neg C \cap \neg D \cap E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B|\neg C \cap \neg D \cap E \cap F) \cdot P(\neg C \cap \neg D \cap E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B|\neg C) \cdot P(\neg C \cap \neg D \cap E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B|\neg C) \cdot P(\neg C|\neg D \cap E \cap F) \cdot P(\neg D \cap E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B|\neg C) \cdot P(\neg C|\neg D) \cdot P(\neg D|E \cap F) \cdot P(E \cap F) \\
&= P(A|\neg B \cap E) \cdot P(\neg B|\neg C) \cdot P(\neg C|\neg D) \cdot P(\neg D) \cdot P(E|F) \cdot P(F) \\
&= 0.2 \cdot 0.7 \cdot 0.5 \cdot 0.9 \cdot 0.3 \cdot 0.2 \\
&= 0.378\%
\end{aligned}
$$

Scenario 9 Probability of compromising the mac algorithm by brute force attack.

$= P(G \cap H \cap \neg I)$

$= P(G|H \cap \neg I) \cdot P(H \cap \neg I)$

$= P(G|H \cap \neg I) \cdot P(H) \cdot P(\neg I)$

$= 0.2 \cdot 0.2 \cdot 0.8$

$= 3.2\%$

Scenario 10 Probability of compromising the mac algorithm by compromising the agent.

$= P(G \cap \neg H \cap I)$

$= P(G|\neg H \cap I) \cdot P(\neg H \cap I)$

$= P(G|\neg H \cap I) \cdot P(\neg H) * P(I)$

$= 0.2 \cdot 0.8 \cdot 0.8$

$= 12.8\%$

Scenario 11 Probability of stopping agent propagation to next LAN by compromising a host at next LAN then poisoning the ARP cache of the LAN router to point to the compromised machine and send back a fake `Seawave_SelfReplicate_ACK` packet to stop the agent from propagating to the target LAN.

$= P(J \cap P \cap K \cap L)$

$= P(J|P \cap K \cap L) * P(P \cap K \cap L)$

$= P(J|P) \cdot P(P \cap K \cap L)$

$= P(J|P) \cdot P(P|K \cap L) \cdot P(K \cap L)$

$= P(J|P) \cdot P(P|K) \cdot P(K|L) \cdot P(L)$

$= 0.6 \cdot 0.5 \cdot 0.2 \cdot 0.2$

$= 1.2\%$

Scenario 12 Probability of capturing the agent code by compromising a host machine and tricking the agent to choose that machine using MAC address flooding.

$= P(M \cap N \cap O \cap \neg Q)$

$= P(M|N \cap O \cap \neg Q) \cdot P(N \cap O \cap \neg Q)$

$= P(M|N) \cdot P(N|O \cap \neg Q) \cdot P(O \cap \neg Q)$

$= P(M|N) \cdot P(N|O \cap \neg Q) \cdot P(O)P(\neg Q)$

$= P(M|N) \cdot P(N|O) \cdot P(O)$

$= 0.5 \cdot 0.5 \cdot 0.2 \cdot 0.7$

$= 3.5\%$

A.1.3 Goal: Compromise Agent to Switch Communication

Scenario 13 Probability of redirecting an agent to probe a rogue host by compromising a host machine and flooding the CAM table with the target mac address.

$= P(A \cap B \cap I \cap \neg J)$

$= P(A|B \cap I \cap \neg J) \cdot P(B \cap I \cap \neg J)$

$= P(A|B) \cdot P(B|I \cap \neg J) \cdot P(I)P(\neg J)$

$= 0.6 \cdot 0.5 \cdot 0.2 \cdot 0.7$

$= 4.2\%$

Scenario 14 Probability of forming a DoS attack by manipulating STP next bridge field.

$= P(C \cap D)$

$= P(C|D) \cdot P(D)$

$= 0.6 \cdot 0.7$

$= 42\%$

Scenario 15 Probability of stopping agent propagation by pointing STP next bridge to non-existent switch and flood CAM table with non-existent MAC.

= $P(E \cap F)$

= $P(E|F) \cdot P(F)$

= $0.6 \cdot 0.7$

= 42%

Scenario 16 Probability of transferring the agent to next LAN by modifying STP next bridge to point to router.

= $P(G \cap H)$

= $P(G|H) \cdot P(H)$

= $0.6 \cdot 0.7$

= 42%

A.1.4 Goal: Compromise Agent to Host Communication

Scenario 17 Probability of capturing exploit code used by agent to probe for vulnerability on a compromised host connected to the same switch.

= $P(A \cap B \cap \neg H)$

= $P(A|B \cap \neg H) \cdot P(B \cap \neg H)$

= $P(A|B \cap \neg H) \cdot P(B) \cdot P(\neg H)$

= $0.5 \cdot 0.2 \cdot 0.7$

= 7%

Scenario 18 Probability of stopping the agent from probing hosts connected to the same switch for vulnerabilities by flooding hosts with `Seawave_Host_Probe_ACK` packets.

$= P(C \cap D \cap I)$

$= P(C|D \cap I) \cdot P(D \cap I)$

$= P(C|D \cap I) \cdot P(D|I) \cdot P(I)$

$= 0.5 \cdot 0.5 \cdot 0.3$

$= 7.5\%$

Scenario 19 Probability of stopping the agent from propagating to the next switch by flooding hosts with `Seawave_SelfReplicate_ACK` packets

$= P(E \cap F \cap \neg G)$

$= P(E|F \cap \neg G) \cdot P(F \cap \neg G)$

$= P(E|F \cap \neg G) \cdot P(F|\neg G) \cdot P(\neg G)$

$= P(E|F \cap \neg G) \cdot P(F) \cdot P(\neg G)$

$= 0.5 \cdot 0.5 \cdot 0.5$

$= 12.5\%$

Scenario 20 Probability of stopping the agent from propagating to the next switch by sending a unicast `Seawave_SelfReplicate_ACK` packet to the agent.

$= P(E \cap \neg F \cap \neg J \cap G \cap K \cap L)$

$= P(E|\neg F \cap \neg J \cap G \cap K \cap L) \cdot P(\neg F \cap \neg J \cap G \cap K \cap L)$

$= P(E|\neg F \cap G) \cdot P(\neg F|\neg J \cap G \cap K \cap L) \cdot P(\neg J \cap G \cap K \cap L)$

$= P(E|\neg F \cap G) \cdot P(\neg F|\neg J) \cdot P(\neg J) \cdot P(G|K) \cdot P(K|L) \cdot P(L)$

$= 0.4 \cdot 0.5 \cdot 0.5 \cdot 0.5 \cdot 0.2 \cdot 0.5$

$= 0.5\%$

A.1.5 Goal: Compromise Agent to Router Communication

Scenario 21 Probability of stopping the agent from propagating on the backbone by feeding malformed `Seawave_LSD_ARP_RPL` (LSD point to no further routers) after compromising OSPF.

$$\begin{aligned}
&= P(A \cap B \cap C \cap \neg D \cap \neg E)\\
&= P(A|B \cap C \cap \neg D \cap \neg E) \cdot P(B \cap C \cap \neg D \cap \neg E)\\
&= P(A|B \cap \neg D) \cdot P(B|C \cap \neg D \cap \neg E) \cdot P(C \cap \neg D \cap \neg E)\\
&= P(A|B \cap \neg D) \cdot P(B|C) \cdot P(C|\neg D \cap \neg E) \cdot P(\neg D \cap \neg E)\\
&= P(A|B \cap \neg D) \cdot P(B|C) \cdot P(C) \cdot P(\neg D|\neg E) \cdot P(\neg E)\\
&= 0.5 \cdot 0.4 \cdot 0.4 \cdot 0.5 \cdot 0.6\\
&= 2.4\%
\end{aligned}$$

Scenario 22 Probability of redirecting or stopping the agent from propagating on the backbone by feeding malformed `Seawave_ARP_RPL` (ARP Table information)

$$\begin{aligned}
&= P(A \cap \neg B \cap \neg C \cap D \cap E)\\
&= P(A|\neg B \cap \neg C \cap D \cap E) \cdot P(\neg B \cap \neg C \cap D \cap E)\\
&= P(A|\neg B \cap D) \cdot P(\neg B|\neg C \cap D \cap E) \cdot P(\neg C \cap D \cap E)\\
&= P(A|\neg B \cap D) \cdot P(\neg B|\neg C) \cdot P(\neg C|D \cap E) \cdot P(D \cap E)\\
&= P(A|\neg B \cap D) \cdot P(\neg B|\neg C) \cdot P(\neg C) \cdot P(D|E) \cdot P(E)\\
&= 0.5 \cdot 0.6 \cdot 0.6 \cdot 0.5 \cdot 0.4\\
&= 3.6\%
\end{aligned}$$

Scenario 23 Probability of an attacker impersonating a router.

$$\begin{aligned}
&= P(F \cap I \cap J)\\
&= P(F|I \cap J) \cdot P(I \cap J)\\
&= P(F|I) \cdot P(I| \cap J) \cdot P(J)\\
&= 0.5 \cdot 0.5 \cdot 0.6\\
&= 15\%
\end{aligned}$$

A.1.6 Goal: Unauthorized Modification of Agent Code

Scenario 24 Probability of Compromising the private key by compromising the master node itself.

$= P(A \cap B \cap \neg C)$

$= P(A|B \cap \neg C) \cdot P(B \cap \neg C)$

$= P(A|B \cap \neg C) \cdot P(B) \cdot P(\neg C)$

$= 0.5 \cdot 0.3 \cdot 0.9$

$= 13.5\%$

Scenario 25 Probability of revealing the private key by brute-force.

$= P(A \cap \neg B \cap C)$

$= P(A|\neg B \cap C) \cdot P(\neg B \cap C)$

$= P(A|\neg B \cap C) \cdot P(\neg B) \cdot P(C)$

$= 0.2 \cdot 0.7 \cdot 0.1$

$= 1.4\%$

A.1.7 Goal: Compromising Agent in Host Machine

Scenario 26 Probability of Compromising the Agent installed in a vulnerable machine.

$= P(A \cap B \cap C)$

$= P(A|B \cap C) \cdot P(B \cap C)$

$= P(A|B) \cdot P(B \cap C)$

$= P(A|B) \cdot P(B|C) \cdot P(C)$

$= 0.4 \cdot 0.5 \cdot 0.5$

$= 10\%$

A.1.8 Goal: Mechanism Information Gathering

Scenario 27 Probability of a malicious user accessing enterprise network and start sniffing agent traffic to allocate master node.

$= P(A \cap B \cap C)$

$= P(A|B \cap C) \cdot P(B \cap C)$

$= P(A|B) \cdot P(B|C) \cdot P(C)$

$= 0.7 \cdot 0.5 \cdot 0.7$

$= 24.5\%$

A.2 Multiple Attack Scenario Probability

A.2.1 Goal: Put The Mechanism Propagation Process into Halt

Multiple Scenarios 28 Probability of a group of adversaries gathering information about the mechanism's traffic, while trying to block the agents from propagating within the switch domain, in addition to disallowing the mechanism from traversing the backbone, forcing the propagation into halt.

$= P(S_{15} \cap S_{21} \cap S_{27})$

$= P(S_{15}|S_{21} \cap S_{27}) \cdot P(S_{21} \cap S_{27})$

$= P(S_{15}|S_{27}) \cdot P(S_{21}|S_{27}) \cdot P(S_{27})$

$= 0.6 \cdot 0.4 \cdot 0.147$

$= 3.528\%$

Bibliography

[1] IEEE Standards for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges (Incorporates IEEE 802.1t-2001 and IEEE 802.1w), 2004. 71, 74, 75

[2] AITEL, D. Nematodes–beneficial worms. *Black Hat Federal* (2006). 33, 39, 44, 47, 55

[3] AL-SALLOUM, Z., AND WOLTHUSEN, S. Semi-Autonomous Link Layer Vulnerability Discovery and Mitigation Dissemination. In *Fifth International Conference on IT Security Incident Management and IT Forensics, (IMF'09).* (Stuttgart, Germany, 2009), pp. 41–53. 27, 39, 61

[4] AL-SALLOUM, Z., AND WOLTHUSEN, S. A Link-Layer-Based Self-Replicating Vulnerability Discovery Agent. In *proceedings of the Fifteenth IEEE Symposium on Computer and Communications (ISCC).* (Riccione, Italy, 2010), pp. 704–707. 27, 39, 62

[5] AL-SALLOUM, Z., AND WOLTHUSEN, S. Agent-Based Host Enumeration and Vulnerability Scanning Using Dynamic Topology Information. In *the 9th Conference of Information Security for South Africa (ISSA)* (Johannesburg, South Africa, Jan 2010). Available from: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5588317`. 27, 62

[6] AL-SALLOUM, Z., AND WOLTHUSEN, S. Security and Performance Aspects of an Agent-Based Link-Layer Vulnerability Discovery Mech-

anism. In *ARES'10 International Conference on Availability, Reliability, and Security – SecSe* (Krakow, Poland, 2010), pp. 549–554. 27

[7] AL-SALLOUM, Z., AND WOLTHUSEN, S. A Propagation Model of A Vulnerability Mitigation Computer Worm - Seawave. In *The Fifth International Conference on Network and System Security.* (Milan, Italy, 2011), IEEE Digital Library. 27

[8] AL-SALLOUM, Z., AND WOLTHUSEN, S. Threat Analysis Model of an Agent-Based Vulnerability Mitigation Mechanism Using Bayesian Belief Networks. In *the International Workshop on Network Science (NSW)* (New York, USA, 2011), IEEE Computer Society. 27

[9] ASHFORD, W. Rsa europe 2010: Botnets have become backbone for cybercrime, says microsoft, 2010. [Online; accessed 4-April-2011]. Available from: `http://www.computerweekly.com/Articles/2010/10/14/243341/RSA-Europe-2010-Botnets-have-become-backbone-for-cybercrime htm`. 29

[10] AYCOCK, J., AND MAURUSHAT, A. Good worms and human rights. *ACM SIGCAS Computers and Society 38*, 1 (2008), 28–39. 60

[11] BAILEY, M., COOKE, E., JAHANIAN, F., WATSON, D., AND NAZARIO, J. The blaster worm: Then and now. *Security & Privacy, IEEE 3*, 4 (2005), 26–31. 30, 38

[12] BELLAMY, L., HUTCHINSON, D., AND WELLS, J. User perceptions and acceptance of benevolent worms–a matter of fear? In *6th International Conference on Computer and Information Science.* (2007), IEEE Computer Society, pp. 11–18. 33, 47, 48

[13] BERBAR, A., AND AHMEDNACER, M. Testing and fault tolerance approach for distributed software systems using nematode worms. *Pro-*

ceedings of the 4th International Conference on Queueing Theory and Network Applications (2009), 1–4. 60

[14] BONTCHEV, V. Are 'good' computer viruses still a bad idea? *vxheavens.com*. Available from: `http://vxheavens.com/lib/avb02.html`. 36, 43

[15] BREITBART, Y., GAROFALAKIS, M., JAI, B., MARTIN, C., RASTOGI, R., AND SILBERSCHATZ, A. Topology discovery in heterogeneous ip networks: the netinventory system. *IEEE/ACM Trans. Netw. 12*, 3 (2004), 401–414. 74, 81, 90

[16] BROWN, R., GRIFFIN, J., NORMAN, A., AND SMITH, R. Why hp did not get "blastered". *HP Laboratories Technical Report HPL-2004-188, Bristol, UK* (2004). 31, 33

[17] BRUCE SCHNEIER. A good worm is hard to find., 2004. [Online; accessed 10-March-2011]. Available from: `http://www.csoonline.com/article/219607/a-good-worm-is-hard-to-find`. 42

[18] CASTANEDA, F., SEZER, E., AND XU, J. Worm vs. worm: preliminary study of an active counter-attack mechanism. *Proceedings of the 2004 ACM workshop on Rapid malcode* (2004), 83–93. 30, 39, 42, 53, 58, 62, 63, 64, 65, 158

[19] CHOI, H., LEE, H., AND KIM, H. Fast detection and visualization of network attacks on parallel coordinates. *Computers & Security 28* (2009). Article in press. 126

[20] CHULANI, S., BOEHM, B., AND STEECE, B. Bayesian analysis of empirical software engineering cost models. *IEEE Trans. Softw. Eng. 25*, 4 (1999), 573–583. 133

[21] COMER, D. E. *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architectures, Fifth Edition*, 5th ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. 70

[22] COMPUTER ECONOMICS. 2007 malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code, 2007. [Online; accessed 21-March-2011]. Available from: `http://www.computereconomics.com/page.cfm?name=Malware%20Report`. 29

[23] CONFICKER WORKING GROUP. Conficker Working Group: Lessons Learned. Tech. rep., Conficker Working Group – confickerworkinggroup.org, Jan 2011. 30, 41

[24] CYRUS PEIKARI. Fighting Fire with Fire: Designing a "Good" Computer Virus, 2004. [Online; accessed 10-March-2011]. Available from: `http://www.informit.com/articles/printerfriendly.aspx?p=337309`. 54

[25] DALEY, D. J., AND GANI, J. M. *Epidemic Modelling : An Introduction*. Cambridge University Press, 1999. 160

[26] FAN, X., AND XIANG, Y. Accelerating the Propagation of Active Worms by Employing Multiple Target Discovery Techniques. In *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC 2008)*, Springer-Verlag. 66

[27] FENTON, N., KRAUSE, P., AND NEIL, M. Software measurement: Uncertainty and causal modeling. *IEEE Softw. 19*, 4 (2002), 116–122. 133

[28] FENTON, N., MARSH, W., NEIL, M., CATES, P., FOREY, S., AND TAILOR, M. Making resource decisions for software projects. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*

(Washington, DC, USA, 2004), IEEE Computer Society, pp. 397–406. 133

[29] FERBRACHE, D. *A Pathology of Computer Viruses*, 1st ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992. 37

[30] FERRIE, P., PERRIOT, F., AND SZOR, P. Worm wars. *VIRUS BULLETIN* (2003). 62

[31] FINLAYSON, MANN, MODUL, THEIMER. A reverse address resolution protocol. *RFC 903, Computer Science Department, Stanford University* (Juni 1984). 80

[32] FOROUGHI, F. Information security risk assessment by using bayesian learning technique. *Proceedings of the World Congress on Engineering 1* (2008). 133

[33] FREIRE, M., FREIRE, M., AND PEREIRA, M. *Encyclopedia of Internet Technologies and Applications*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2007. 35

[34] GILES HOGBEN. Botnets: Detection, Measurement, Disinfection and Defence. Tech. rep., European Network and Information Security Agency (ENISA), Mar 2011. 29, 46

[35] GUPTA, A., AND DUVARNEY, D. Using predators to combat worms and viruses: A simulation-based study. *20th Annual Computer Security Applications Conference* (2004), 116–125. 56

[36] H. SEYBERT, A. LF. Internet usage in 2010 Households and Individuals. Tech. rep., Eurostat, 2011. 29

[37] HANS DELFS, H. K. *Introduction to Cryptography Principles and Applications*. Springer-Verlag, Berlin Heidelberg New York, 2007. 110

[38] HARRINGTON, D., PRESUHN, R., AND WIJNEN, B. An architecture for describing simple network management protocol (snmp) management frameworks, 2002. 70

[39] HECKERMAN, D. A tutorial on learning with bayesian networks. Tech. rep., Learning in Graphical Models, 1995. 132

[40] IEEE. IEEE Standard for Local and metropolitan area networks Station and Media Access Control Connectivity Discovery. *"IEEE Std 802.1AB2005"* (2005), 0_1-158. 106

[41] II, J. W. C. *The Oxford Companion to American Military History*, 1st ed. Oxford University Press, USA; 1st edition, Cary, NC, USA, 2000. 47

[42] ISSARIYAKUL, T., AND HOSSAIN, E. *Introduction to Network Simulator NS2*. Springer-Verlag, Heidelberg, Germany, 2009. 79, 112, 123, 174

[43] JAMES MIDDLETON. 'Anti-worms' fight off Code Red threat, 2001. [Online; accessed 10-March-2011]. Available from: `http://www.v3.co.uk/v3-uk/news/1962686/anti-worms-fight-code-red-threat`. 63, 64

[44] JENSEN, F. V., AND NIELSEN, T. D. *Bayesian Networks and Decision Graphs*. Springer Publishing Company, Incorporated, 2007. 148

[45] JODIE NAZE. Beneficial Network Worms, 2006. [Online; accessed 10-March-2011]. Available from: `http://www.itworld.com/nls_security_nematodes_060117`. 47

[46] JOHN LEYDEN. Blaster variant offers 'fix' for pox-ridden pcs, 2003. [Online; accessed 10-March-2011]. Available from: `http://www.theregister.co.uk/2003/08/19/blaster_variant_offers_fix/`. 62

[47] JUDGE HOWARD G. MUNSON. 928 F.2D 504: United States of America v. Robert Tappan Morris, 1991. Available from: `http://morrisworm.larrymcelhiney.com/morris_appeal.txt`. 37

[48] JUNG, J., MILITO, R. A., AND PAXSON, V. On the Adaptive Real-Time Detection of Fast-Propagating Network Worms. *Journal of Computer Virology 4*, 3 (Aug. 2008), 197–210. 126

[49] KASPERSKY. Net-Worm.Perl.Santy.a threatens Internet Forums, 2004. [Online; accessed 16-April-2011]. Available from: `http://www.kaspersky.co.uk/news?id=156681162`. 65

[50] KERN, M. Re: Codegreen beta release (idq-patcher/anticodered/etc.), 2001. [Online; accessed 13-April-2010]. Available from: `http://seclists.org/vuln-dev/2001/Sep/0001.html`. 67

[51] KHAYAM, S., AND RADHA, H. A topologically-aware worm propagation model for wireless sensor networks. *25th IEEE International Conference on Distributed Computing Systems Workshops.* (2005), 210–216. 55

[52] KONDAKCI, S. Network Security Risk Assessment Using Bayesian Belief Networks. *The 2nd IEEE International Conference on Information Privacy, Security, Risk and Trust* (2010). Available from: `ftp://pubftp.computer.org/Press/outgoing/proceedings/SocialCom/data/4211a952.pdf`. 133, 134, 152

[53] LEDER, F., AND WERNER, T. Know Your Enemy: Containing Conficker. Tech. rep., The Honeynet Project, University of Bonn, Germany, Mar. 2009. 67, 121

[54] LEMOS, R. Counting the cost of slammer, 2003. [Online; accessed 24-March-2011]. Available from: `http://news.cnet.com/2100-1001-982955.html`. 29

[55] LILJENSTAM, M., AND NICOL, D. M. Comparing Passive and Active Worm Defenses. In *Proceedings of the First International Conference on the Quantitative Evaluation of Systems (QEST)* (September 2004), pp. 18–27. 53

[56] LIU, Y., YUN, X., WANG, B., AND SUN, H. An Anti-worm with Balanced Tree Based Spreading Strategy. *Advances in Machine Learning and Cybernetics* (2006), 652–661. 57

[57] MARK JOSEPH EDWARDS. Code red turns codegreen, 2001. [Online; accessed 10-March-2011]. Available from: `http://www.windowsitpro.com/article/antivirus/code-red-turns-codegreen.aspx`. 63, 64

[58] MARSON, I. Anti-santy worm on the prowl, 2004. [Online; accessed 16-April-2011]. Available from: `http://news.cnet.com/2100-7349_3-5508607.html`. 65

[59] THE MATHWORKS, INC. *MATLAB Getting Started Guide*, 2010. 171

[60] MCAFEE LABS. 2011 Threats Predictions. Tech. rep., McAfee, Inc., Jan. 2011. 29

[61] MCMILLAN, R. Dhs chief: What we learned from stuxnet, 2011. [Online; accessed 27-April-2011]. Available from: `http://www.pcworld.com/businesscenter/article/226237/dhs_chief_what_we_learned_from_stuxnet.html`. 23, 46

[62] MICROSOFT. Microsoft Security Intelligence Report volume 9 Battling Botnets. 1–70. 29

[63] MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S., AND WEAVER, N. Inside the Slammer Worm. *IEEE Security and Privacy 1*, 4 (July 2003), 33–39. 30, 38, 77, 78

[64] MOORE, D., AND SHANNON, C. The Spread of the Code-Red Worm (CRv2), 2001. Available from: `http://www.caida.org/research/security/code-red/coderedv2_analysis.xml`. 158

[65] MOORE, D., AND SHANNON, C. The spread of the Witty worm. *IEEE security and privacy 2*, 4 (2004). 38

[66] MOORE, D., SHANNON, C., AND BROWN, J. Code-red: A Case Study on the Spread and Cictims of an Internet Worm. In *Proceedings of the ACM SIGCOMM / USENIX Internet Measurement Workshop (IMW 2002)* (Marseille, France, 2002), pp. 273–284. 29, 30, 38, 66

[67] MOY, J. T. The OSPF Specification. Internet Request For Comments RFC 1131, Internet Engineering Task Force, Oct. 1989. 88

[68] MOY, J. T. OSPF Version 2. Internet Request For Comments RFC 1247, Internet Engineering Task Force, July 1991. 88

[69] MOY, J. T. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998. 90

[70] N FALLIERE AND L.O MURCHU AND E CHIEN. W32. Stuxnet Dossier. Tech. rep., Symantec Security Response, 2011. 30, 41

[71] NAZARIO, J. *Defense and Detection Strategies against Internet Worms*. Artech House Publishers, October 2003. Available from: `Hardcover`. 30, 37, 64, 65, 66

[72] NAZARIO, J., ANDERSON, J., WALSH, R., AND CONNELLY, C. The Future of Internet Worms. Tech. rep., Crimelabs Research, 2001. 93, 110

[73] NAZARIO, J., PTACEK, T., AND SONG, D. Wormability: A description for vulnerabilities. *Arbor Networks* (Jan 2004). Avail-

able from: http://98.15.203.119/eBooks/Virusability_researchOct04.pdf. 32

[74] NICOL, D., AND LILJENSTAM, M. Models of Active Worm Defenses. *IPSI Conference 51* (2004), 61801. 52

[75] NIE, X., AND JING, J. A Novel Contagion-Like Patch Dissemination Mechanism against Peer-to-Peer File-Sharing Worms. *Information Security and Cryptology* (Jan 2011). Available from: http://www.springerlink.com/index/75M310572822351K.pdf. 61

[76] NORSYSSOFTWARE CORP. Netica Application. Available from: http:/www.norsys.com. 154

[77] PARSONS, J. J., AND OJA, D. *New Perspectives on Computer Concepts 2011: Introductory*, 13th ed. Course Technology Press, Boston, MA, United States, 2010. 35

[78] PENDHARKAR, P. C., SUBRAMANIAN, G. H., AND RODGER, J. A. A probabilistic model for predicting software development effort. In *ICCSA'03: Proceedings of the 2003 international conference on Computational science and its applications* (Berlin, Heidelberg, 2003), Springer-Verlag, pp. 581–588. 132, 133

[79] PHILLIPS, C., AND SWILER, L. P. A Graph-Based System for Network-Vulnerability Analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms* (New York, NY, USA, 1998), ACM, pp. 71–79. 134

[80] PLUMMER, D. C. *RFC 0826 Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*, November 1982. 83

[81] SCARFONE, K., GRANCE, T., AND MASONE, K. Computer Security Incident Handling Guide. *NIST Special Publication 800* (2008), 61. 38

[82] SCHNEIER, B. Gathering 'storm' superworm poses grave threat to pc nets. *Wired.com* (2007). [Online; accessed 09-May-2011]. Available from: `http://www.wired.com/politics/security/commentary/securitymatters/2007/10/securitymatters_1004`. 41

[83] SEAGREN, E. *Secure Your Network for Free*. Syngress Publishing, 2007. 78

[84] SEBASTIAN BORTNIK. Conficker by the numbers. Tech. rep., ESET Latin America, Feb 2010. 29

[85] SEBORG, B. An article in the electronic forum VirusL/comp.virus. 35

[86] SEIFERT, R., AND EDWARDS, J. *The All-New Switch Book: The Complete Guide to LAN Switching Technology*. Wiley Publishing, 2008. 73, 75, 92

[87] SHOCH, J., AND HUPP, J. The "worm" programs—early experience with a distributed computation. *Communications of the ACM 25*, 3 (1982), 172–180. 50

[88] SIMONITE, T. Friendly 'worms' could spread software fixes. *New-Scientist* (2008). [Online; accessed 13-April-2011]. Available from: `http://www.newscientist.com/article/dn13318`. 66

[89] SKOUDIS, E., AND ZELTSER, L. *Malware: Fighting Malicious Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003. 31

[90] SOLOMON, A. *PC Viruses: detection, analysis, and cure*. Springer-Verlag New York, Inc., New York, NY, USA, 1991. 35

[91] SPAFFORD, E. The Internet Worm Incident Technical Report csd-tr-933. *Lecture Notes in Computer Science87, Springer-Verlag* (1989). 37, 38

[92] SPAFFORD, E. H. The Internet Worm Program: An Analysis. Tech. Rep. CSD-TR-823, Department of Computer Sciences, Purdue University, West Lafayette, IN, USA, Dec. 1988. 66

[93] STANIFORD, S., PAXSON, V., AND WEAVER, N. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium* (Berkeley, CA, USA, 2002), USENIX Association, pp. 149–167. 66, 158, 160

[94] STEVEN, J. Threat modeling perhaps it's time. *IEEE Security and Privacy 8*, 3 (2010), 83–86. 131

[95] STOTT, D. T. Layer-2 Path Discovery using Spanning Tree MIBs. *Avaya Labs Research, Avaya Inc., Basking Ridge, NJ, Tech. Rep* (2002), 2002–004. 74, 81, 82

[96] SU, F. Modeling and analysis of internet worm propagation. *The Journal of China Universities of Posts and Telecommunications 17*, 4 (2010), 63–68. 159

[97] SZOR, P. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005. 41, 63

[98] TANACHAIWIWAT, S., AND HELMY, A. VACCINE: War of the Worms in Wired and Wireless Networks. *IEEE International Conference on Computer Communications* (2006), 05–859. 57

[99] TANACHAIWIWAT, S., AND HELMY, A. Modeling and Analysis of Worm Interactions (War of the Worms). *Fourth International Conference on Broadband Communications, Networks and Systems, (BROADNETS 2007).* (2007), 649–658. 58

[100] TANACHAIWIWAT, S., AND HELMY, A. On the Performance Evaluation of Encounter-Based Worm Interactions based on Node Characteristics. *ACM MobiCom Workshop on Challenged Networks* (2007). 58

[101] TOUTONJI, O., AND YOO, S.-M. Passive Benign Worm Propagation Modeling with Dynamic Quarantine Defense. *TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS 3*, 1 (2009), 96–107. 61, 159

[102] TOYOIZUMI, H., AND KARA, A. Predators: Good will Mobile Codes combat against Computer Viruses. *Proceedings of the workshop on New security paradigms* (2002), 11–17. 51, 55

[103] VOJNOVIĆ, M., GUPTA, V., KARAGIANNIS, T., AND GKANTSIDIS, C. Sampling Strategies for Epidemic-Style Information Dissemination. In *Proceedings of the 27th IEEE Conference on Computer Communication (INFOCOM 2008)* (Phoenix, AZ, USA, Apr. 2008), IEEE Press, pp. 1678–1686. 59, 66

[104] WANG, B., FANG, B., AND YUN, X. The Propagation Model and Analysis of Worms Together with Anti-Worms. *WSEAS transactions on information science and applications 4*, 1 (2004), 967–982. 37, 51, 62

[105] WANG, F., SONG, J., DONG, Y., AND GU, J. Epidemic Models Applied to Worms on Internet. *Second International Conference on Intelligent Networks and Intelligent Systems* (2009), 160–163. 60

[106] WANG, H. J., WANG, H. J., GUO, C., GUO, C., SIMON, D. R., SIMON, D. R., ZUGENMAIER, A., AND ZUGENMAIER, A. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *ACM SIGCOMM* (2004), pp. 193–204. 31, 32

[107] WEAVER, N., PAXSON, V., STANIFORD, S., AND CUNNINGHAM, R. A Taxonomy of Computer Worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode* (New York, NY, USA, 2003), WORM '03, ACM, pp. 11–18. Available from: `http://doi.acm.org/10.1145/948187.948190`. 37

[108] WHALLEY, I., ARNOLD, B., CHESS, D., MORAR, J., SEGAL, A., AND SWIMMER, M. An Environment for Controlled Worm Replication and Analysis. *IBM TJ Watson Research Center* (2000). 37

[109] WU, D., LONG, D., WANG, C., AND GUAN, Z. Modeling and Analysis of Worm and Killer-Worm Propagation using the Divide-and-Conquer Strategy. *Distributed and Parallel Computing* (2005), 370–375. 56, 58

[110] YANG, Y., FANG, Y., AND LI, L.-Y. The Analysis of Propagation Model for Internet Worm Based on Active Vaccination. In *Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 06* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 682–688. Available from: `http://portal.acm.org/citation.cfm?id=1473248.1473980`. 159

[111] YAO, Y., LV, J., GAO, F., YU, G., AND DENG, Q. Detecting and Defending against Worm Attacks Using Bot-honeynet. *Second International Symposium on Electronic Commerce and Security* (2009), 260–264. 60

[112] YONG, Y. A New Vaccine for Worm Vaccination. In *Proceedings of the 2010 International Conference on Artificial Intelligence and Computational Intelligence - Volume 01* (Washington, DC, USA, 2010), AICI '10, IEEE Computer Society, pp. 445–447. Available from: `http://dx.doi.org/10.1109/AICI.2010.99`. 61

[113] ZEGURA, E. W. GT-ITM: Georgia Tech Internetwork Topology models (software), 1996. Available from: `http://www.cc.gatech.edu/project`. 104

[114] ZHOU, H., WEN, Y., AND ZHAO, H. Modeling and Analysis of Active Benign Worms and Hybrid Benign Worms Containing the Spread of

Worms. *Sixth International Conference on Networking, ICN'07.* (2007), 65. 59

[115] ZHOU, H., WEN, Y., AND ZHAO, H. Passive Worm Propagation Modeling and Analysis. In *Proceedings of the International Multi-Conference on Computing in the Global Information Technology* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 32–. 58, 159

[116] ZOU, C., GONG, W., AND TOWSLEY, D. Code Red Worm Propagation Modeling and Analysis. *Proceedings of the 9th ACM conference on Computer and communications security* (2002), 138–147. 58, 158

[117] ZOU, C., TOWSLEY, D., AND GONG, W. On the Performance of Internet Worm Scanning Strategies. *Performance Evaluation 63*, 7 (2006), 700–723. 66, 159, 160, 161

[118] ZOU, C. C., TOWSLEY, D., GONG, W., AND CAI, S. Routing Worm: A Fast, Selective Attack Worm Based on IP Address Information. In *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 199–206. 66