8-2007

ENTERPRISE SECURITY ANALYSIS INCLUDING DENIAL OF SERVICE COUNTERMEASURES

Chinar Dingankar
Clemson University, cdingan@clemson.edu

ENTERPRISE SECURITY ANALYSIS INCLUDING DENIAL OF SERVICE COUNTERMEASURES

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Chinar Dingankar
August 2007

Accepted by:
Dr. Richard Brooks, Committee Chair
Dr. Samuel Sander
Dr. Ian Walker

ABSTRACT

Computer networks are the nerve systems of modern enterprises. Unfortunately, these networks are subject to numerous attacks. Safeguarding these systems is challenging. In this thesis we describe current threats to enterprise security, before concentrating on the Distributed denial of Service (DDoS) problem.

DDoS attacks on popular websites like Amazon, Yahoo, CNN, eBay, Buy, and the recent acts of war using DDoS attacks against NATO ally Estonia [1] graphically illustrate the seriousness of these attacks. Denial of Service (DoS) attacks are explicit attempts to block legitimate users' system access by reducing system availability [2]. A DDoS attack deploys multiple attacking entities to attain this goal [3]. Unfortunately, DDoS attacks are difficult to prevent and the solutions proposed to date are insufficient. This thesis uses combinatorial game theory to analyze the dynamics of DDoS attacks on an enterprise and find traffic adaptations that counter the attack.

This work builds on the DDoS analysis in [4]. The approach we present designs networks with a structure that either resists DDoS attacks, or adapts around them. The attacker (Red) launches a DDoS on the distributed application (Blue). Both Red and Blue play an abstract board game defined on a capacitated graph, where nodes have limited CPU capacities and edges have bandwidth constraints. Our technique provides two important results that aid in designing DDoS resistant systems:

1. It quantifies the resources an attacker needs to disable a distributed application. The design alternative that maximizes this value will be the least vulnerable to DDoS attacks.

2. When the attacker does not have enough resources to satisfy the limit in 1, we provide near optimal strategies for reconfiguring the distributed application in response to attempted DDoS attacks.

Our analysis starts by finding the feasible network configurations for Blue that satisfy its computation and communications requirements. The min-cut sets [5] of these configurations are the locations most vulnerable to packet flooding DDoS attacks.

Red places “zombie” processes on the graph that consume network bandwidth. Red attempts to break Blue communications links. Blue reconfigures its network to re-establish communications. We analyze this board game using the theory of surreal numbers [6]. If Blue can make the game “loopy” (i.e. move to one of its previous configurations), it wins [7]. If Red creates a situation where Blue can not successfully reconfigure the network, it wins.

In practice, each enterprise relies on multiple distributed processes. Similarly, an attacker can not expect to destroy all of the processes used by the enterprise at any point in time. The attacker will try to maximize the number of processes it can disable at any point in time. This situation describes a “sum of games” problem [6], where Blue and Red alternate moves. We adapt Berlekamp’s strategies for Go endgames, to tractably find near optimal reconfiguration regimes for this P-Space complete problem [6], [7].

DEDICATION

This thesis is dedicated to *My Parents, Rafika and Fazal, my sisters Tamanna, and Mehjabeen, my brothers Irfan and Mohsin, all my friends, Shivani, Naresh, Neha, Prabhu, Vibhav, Nikhil, Shivang, Satya, Tarik, Ganesh, Sandeep, Haresh, Mahesh, Priyanka, Swetha, Purva* for their loving support and encouragement, and *Aftab,* my partner for life without whom this would not have been possible. I thank *God* for giving me the strength and determination to achieve my goals.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my advisor Dr. Richard R. Brooks for his guidance throughout this research. I am grateful for his support and patience during the entire period of this research. I would like to thank Dr. Samuel Sander, Dr. Ian D. Walker and Dr. Joachim Taiber for serving as my graduate committee members.

The financial support of BMW, ITRC, Greenville and the Electrical and Computer Engineering (ECE) Department at Clemson University are greatly appreciated.

I would like to thank all my colleagues in my research group – Ramki, Neha, Brijesh, Prashanth, Priyanka, Jing-en, Jason, Juan and Hung for their professional and friendly relationship.

Finally I would like to thank my parents for their patience and their emotional and financial support throughout my life.

TABLE OF CONTENTS

Page

TITLE PAGE i

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1. INTRODUCTION 1

1.1 Motivation 1
1.2 DDoS Mitigation 4
1.3 Research Objectives / Contributions 5
1.4 Thesis Outline 5

2. ENTERPRISE SECURITY 7

2.1 Background 8
2.1.1 Assets to be protected with respect to key Players 9
2.1.2 Attackers and their Motivation 9
2.1.3 Attacks 10
2.1.3.1 Input Exploitation 12
2.1.3.2 Code Insertion 16
2.1.3.3 Platform attacks 19
2.1.3.4 Session ID Attacks 25
2.1.4 Attack Trees 27

3. BACKGROUND ON DENIAL OF SERVICE ATTACKS AND EXISTING MITIGATION TECHNIQUES 30

Table of Contents (Continued)

Page

3.1 Denial-of-Service and Distributed-Denial-of-Service Attacks 30
3.2 Related Work in Mitigation of Distributed Denial of Service 32

4. DISTRIBUTED DENIAL OF SERVICE MODEL 36

4.1 Problem Statement for DDoS Games 36
4.1.1 Physical Environment 36
4.1.2 Blue (Virtual) Environment 37
4.2 Key Concepts 39

5. DISTRIBUTED DENIAL OF SERVICE MITIGATION APPROACH – TRAFFIC FLOW 42

5.1 Player 1 - Blue 42
5.1.1 Node Capacity Constraints 43
5.1.2 Edge Capacity Constraints 45
5.1.3 Verification 1 46
5.1.4 Verification 2 47
5.1.5 Verification 3 48
5.1.6 Feasible Blue Configurations 49
5.2 Player 2 - Red 49
5.2.1 Attack Node Capacities 50
5.2.2 Attack Arc Capacities 57
5.2.3 Flooding the Arcs and Zombie Traffic 58
5.2.4 Zombie Placement 61

6. DISTRIBUTED DENIAL OF SERVICE MITIGATION APPROACH – RECONFIGURATION STRATEGIES 63

6.1 Game 63
6.1.1 Surreal Numbers and Combinatorial Game Theory 63
6.1.2 Example Game 68
6.2 Sum of Games and Thermographs 71
6.2.1 Example 80

7. SIMULATIONS AND RESULTS 87

Table of Contents (Continued)

Page

7.1 Results from MATLAB Tool........87
7.2 SSFNet Simulations........88
7.3 Example and Discussion........90

8. CONCLUSIONS AND FUTURE RESEARCH........96

APPENDICES........99

A: Matlab Tool Results........100
B: SSFNet Simulations........105

REFERENCES........112

LIST OF TABLES

Table Page

5.1 Blue configurations and min-cut nodes 56

5.2 Blue configurations and min-cut nodes 58

6.1 Details of an example Game 70

6.2 Right boundaries for compound thermograph at each tax interval for Left (Blue) 78

6.3 Maximum Width at each tax interval 79

6.4 Adding Maximum Width and the Right Boundary of each game 79

6.5 Right boundaries for compound thermograph at each tax interval for Left (Blue)Adding Maximum Width and the Right Boundary of each game 85

6.6 Maximum Width at each tax interval 85

6.7 Adding Maximum Width and the Right Boundary of each game 85

LIST OF FIGURES

Figure Page

1 DDoS Attack 3

2 Computer attack incident taxonomy from [24] used in [4, 24] 12

3 Attack tree to attack the Database 28

4 Attack tree to attack the Web Server / Database using the Buffer Overflow vulnerability 29

5 Attack Max-flow and min-cut for a directed graph structure 41

6 Node Capacity Constraints 44

7 Connectivity Graphs (a) Physical Connectivity Graph (b) Blue Connectivity Graph 46

8 Reducing the number of arcs Red has to disable 51

9 Flow chart for reducing the number of arcs in the min-cut 53

10 Branch and Bound Search Tree for Node Attack 56

11 Branch and Bound Search Tree for Arc Attack 58

12 Game tree representation for G 67

13 Game Trees (a) Red Wins (b) Loopy Game- Blue (NA – Not Allowed) 71

14 Thermograph of the game {2 | -1} 74

15 Thermograph for {{5 | -5} || -20} 75

16 Thermographs for *B* and *A* 77

17 Game Tree for distributed application 1 – Game 1 81

List of Figures (Continued)

Figure Page

18 Game Tree for distributed application 2 – Game 2 82

19 Game Tree for distributed application 3 – Game 3 83

20 Thermographs to calculate sum of games (a) Game 1 (b) Game 2 (c) Game 3 84

21 Input Networks (a) Physical Network (b) Blue Network 87

22 SSFNet animation snapshot showing the colors assigned to different hosts 90

23 SSFNet simulation for Example 1 (a) Animation (b) Graphical statistics 93

24 Graphical statistics of attack traffic of a large network 94

25 SSFNet representation of two Nodes A and B 107

26 SSFNet snapshot showing the three components of graphical representation 111

CHAPTER 1

INTRODUCTION

1.1 Motivation

The Internet has become part of the nation's critical infrastructure. Critical financial services like banking, bill payment, tax payment, booking travel reservations, and shopping require secure communications. These communications frequently use the Internet [8], and the Internet's security flaws expose these services to abuse. To be secure, a system must provide availability, data integrity, confidentiality, non-repudiation, access control and authentication [9]. Weakness in any of these areas can be exploited to undermine system integrity. A secure system should also be able to correctly respond to attacks [4]. A Denial of Service (DoS) attack is an explicit attempt to prevent legitimate users of a service from having access to that service [10]. In a Distributed Denial of Service (DDoS) attack, multiple compromised systems (called zombies) attack a single target. Coordinated DDoS attacks originate from multiple machines simultaneously.

A DoS attack does not usually result in the theft of information. However, these attacks on the can cost the target a great deal of time and money. For example the foreign governments that cannot attack United States on a battle field due to its military dominance could use these low cost, high profile, large effect cyber attacks to cripple its defense. The military and civilian information infrastructures are increasingly intertwined making the Department of Defense (DoD) dependent on the National Information

Infrastructure (NII) for maintaining communications, command control and intelligence capabilities [11]. The military infrastructure has been a victim of cyber attacks. In 1997, a particular red team of hackers was able to penetrate network defenses and take control of the Pacific command center computers, power grids and 911 systems in nine major cities of the United States [12]. Later on in 1998, attacks like Solar Sunrise and Moonlight Maze on the military infrastructure were also reported [4].

The number of these attacks on the Internet has risen sharply in the past several years and the Internet root servers are vulnerable to these attacks [13]. DDoS attacks, where a single attacker (*master*) coordinates many hijacked systems (*zombies*) see Figure 1, are difficult to trace. It is almost impossible to find the source of a DDoS attack, which makes system recovery a slow and costly process. One DDoS attack, which is difficult to trace, is the smurf attack. Smurfing uses spoofed broadcast ping messages to flood a target system; the attacker directs zombies to send Internet Control Message Protocol (ICMP) traffic to a set of Internet Protocol (IP) broadcast addresses. The ICMP echo packets have a spoofed source address pointing to the intended victim. Hosts accept the ICMP ping requests and reply to the source address, in this case the victim. This multiplies the flooding traffic by the number of responding hosts [14]. This also adds an additional layer of anonymity to the DDoS attacks and makes finding the attacker very difficult.

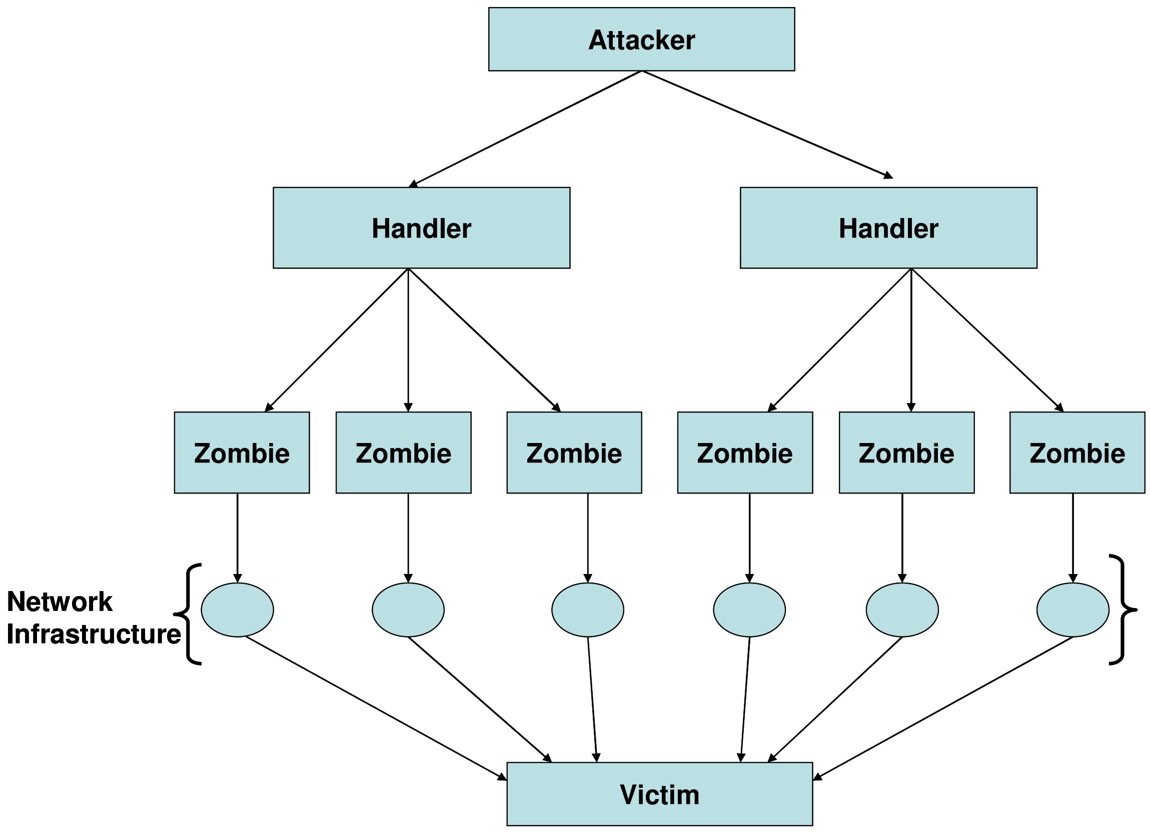


Figure 1. DDoS Attack.

DDoS attacks on popular websites:

- Yahoo, Amazon , CNN, eBay, Buy, etc from February 7th to 11th 2001,
- The White House on May 4th 2001,
- Microsoft on January 24th, 2001, and
- Internet domain name system root servers on October 23rd, 2002

have brought public attention to the seriousness of this kind of security breach [13].

Perhaps the best illustration of this problem is the first war fought totally in cyberspace; Russia's attack on Estonia using DDoS attacks in 2007 [1]. The DDoS attack on Estonia continued for three weeks. The e-mail server of the Estonian Parliament and the online service of the biggest bank in Estonia were both incapacitated. This information war cost Estonia millions of dollars. In response to these attacks, Estonia reacted by filtering attack packets. But the authorities have yet to identify the attacking computers, thanks to the attacker's ability to mask (spoof) the IP addresses of participating computers. The attackers used about one million compromised computers (zombies) in places as far away as the United States and Vietnam to amplify their attack.

The owners of zombie computers are usually unaware their machines are participating in a DDoS attack. This, and the indirect nature of the attack shown in Figure 1, makes it almost impossible to find the source of an attack. Finding the attack source is necessary not only for identifying the attackers, but also for stopping the DDoS attack.

Estonia's business sector came to a stand still but all it could do in retaliation was filter attack packets. Was there no other way of mitigating these attacks? To thwart these attacks we either need to build robust networks that can sustain DDoS attacks or find mitigation strategies that stop DDoS attacks before damage is done.

1.2 DoS Mitigation

Most proposed defenses against DDoS attacks are reactive (attack-specific) and involve identifying attack source(s) and deactivating them [15]. Other countermeasures involve network monitoring: ingress filtering, egress filtering, intrusion detection, etc. Safeguarding a network is good. But rather than waiting for an enemy to attack, it's better to defend a network by eliminating vulnerabilities.

1.3 Research Objectives / Contributions

Our work analyzes a two player game played on a computer network. The computer network is modeled by a directed graph structure in which computers are denoted by nodes and links connecting computers are represented by arcs. Player 1 (Blue) is a distributed application on the network. Player 2 (Red) is an attacker that places zombie processes on network nodes to either:

1. Attack node capacities.
2. Flood arcs.

We look for the minimum number of zombies required to disable Blue's distributed system. The smallest set of zombies needed to disable all Blue's possible configurations quantifies the resistance of Blue to DDoS attacks. If the attacker does not have enough zombies to disable all Blue configurations, Blue can reconfigure to recover from the DDoS attack. This defines a simple board game. Red tries to force Blue into a position where it cannot recover by reconfiguring. We use combinatorial game theory to analyze this problem. The results we obtain can be used when designing robust networks in the future.

1.4 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 surveys the security measures currently used for maintaining enterprise security. Chapter 3 gives background on DoS attacks and existing mitigation techniques. Chapter 4 explains how we model the network and the key concepts behind our approach. Chapter 5 and chapter 6 explain our approach to attack mitigation based on traffic flow and process reconfiguration strategies, respectively. Chapter 7 presents simulation results. We conclude the thesis with Chapter 8 presenting our conclusions and future directions for research.

CHAPTER 2

ENTERPRISE SECURITY

Corporations have become increasingly aware of the importance of computer network security. Until recently, most companies concentrated only on making their network available to users at all times. Network security was seen as an unnecessary additional cost. The increased number of attacks and financial losses have forced corporations to take security seriously [16], [17], [18], [19].

Perfect security is widely acknowledged to be a myth [20]. Real world security is a system of trade-offs. Corporations try to make system intrusions more expensive for the attacker to execute than the attacker can expect to gain. At the same time, corporations must verify that their security expenditures are spent wisely, so that they minimize their own expected loss. To discuss current enterprise security, we present the following:

1. Description of system stakeholders, the assets they need to protect, and the value of those assets.

2. Taxonomy of potential attacks based on the approach used by the Computer Emergency Response Team (CERT) is given to describe and categorize system intrusions. In this context, it allows one to consider classes of attacks rather than individual instances.

3. Attack trees are diagrams representing how the most likely, and potentially damaging, network intrusions could occur. Security auditors (red teams) commonly use attack trees to analyze infrastructure security. In the tree, an attack is decomposed into a sequence of and/or conditions.

4. System penetration testing finds vulnerabilities in the current system. System penetration testing should be done on a test environment when the system is in production or development. Once the system is launched the penetration testing should be done at regular intervals to find new vulnerabilities and test for new

attacks. This includes performing an exhaustive test of data tampering attacks on user input fields.

2.1. Background

This section provides relevant background information on the users of a system, other key players involved, the assets that need to be protected. It is important to identify what each individual has at stake and the potential losses that can occur. It is also important to identify the attackers, their estimated gains from attacking a system, the potential vulnerabilities of a system and the entry point for an attack. To properly prioritize threats, it is important for an enterprise to realize,

1. The users and the enterprise,
2. Other key players involved,
3. Assets that each player wants to protect,
4. How the system is supposed to be used,
5. The potential attackers of the system,
6. The motivation of the attackers, and
7. System vulnerabilities,
8. Attack entry points.

For further information [16] offers a good review of enterprise security measures.

2.1.1 Assets to be protected with respect to Key Players

The assets that need to be protected with respect to each player include

1. Enterprise – needs to maintain the security of the system in a way that secures its intellectual property, reputation and freedom from liability litigation. It also needs to guarantee the correct functioning of the network to keep it available and guard against improper use of the applications. The duties include maintaining the traditional security attributes of confidentiality, authentication, integrity, non-repudiation, access control, and availability [4], [9].

2. Users – need to protect their personal data. The owner also wants to maintain their system in working order. Typically customers want to guard against the abuse of their user account. They are particularly sensitive to account abuse that violates their expectations of privacy and data integrity. Each customer needs to safeguard their authentication credentials from exposure to potential attackers. It is worth noting that the assumption that users adequately safeguard their private information is often the weak link in security plans. This is due both to the ease of constructing social engineering attacks [21] and the unfortunate fact that theft of personal data is frequently done by family or friends [22]. Recent better business bureau statistics even show 47% of identity theft is performed by "friends, neighbors, in-home employees, family members, or relatives" [23].

3. Other players (partners, suppliers, vendors) - are interested in safeguarding themselves from liability litigation and maintaining their reputations. They also want to safeguard shared resources like network, information access to prevent potential legal liabilities.

2.1.2. Attackers and their Motivation

The attackers that will attack an enterprise typically belong to one or more of the following classes [4], [24], [25],

1. Hackers – are typically individual computer users with technical literacy skills. Stereotypically, hackers are motivated by their interest in technology and the

intellectual challenge of overcoming security measures. Recent studies indicate that this type of computer abuse is waning. The profit motive seems to be the prime motivator in recent computer security incidents, which means that this class of attackers and the criminal class are becoming more difficult to differentiate.

2. Corporate raiders – increasingly use technical means for performing industrial espionage. Industrial espionage may be limited to discovering trade secrets or may include sabotage to disrupt the services provided by a competitor. In many countries, the government and industry are intertwined, so that corporate espionage may be performed by the espionage services of nation states [26], [27]. This class of attackers typically has the most resources to invest in an attack.
3. Criminals – have discovered a number of methods for making money dishonestly on-line. Criminal elements are involved in identity theft, spamming, and other fraudulent transactions. This may involve hijacking multiple machines using malware and coordinating their activities to form a "botnet." There are many documented cases of botnets being used to create Distributed Denial of Service (DDoS) attacks as part of a protection racket. In many regions organized criminal gangs are actively involved in trafficking stolen luxury automobiles.
4. Legitimate users – are likely to try to steal services of an enterprise. The theft of service attacks will occur, where legitimate users will try to access costly services without paying for them. A few disgruntled customers may also attempt to sabotage the services of an enterprise with a revenge motive.
5. Enterprise corporate insiders – are very likely to abuse their system access rights. Disgruntled or bored workers may try to sabotage the network. Workers may also be bribed by criminals or competitors to commit unethical acts. Recent studies rank the insider threat as the second most prevalent source of computer crime [28], [29]. Since these users have legitimate access needs, and may even be in charge of enforcing security regulations, it is particularly difficult to guard the system against malicious insiders.

These attackers have varying motivations from interest in computer systems to revenge for perceived wrongs to monetary gain.

2.1.3. Attacks

Attacks are rarely novel. In order to evaluate the security of an enterprise the classes of attacks that can be possible are discussed. Further the common attacks that can be hazardous to an enterprise are discussed in detail.

Figure 2 shows an attack incident taxonomy used by the Computer Emergency Response Team (CERT) to describe security incidents [4], [24], [25]. This taxonomy provides a useful framework and uniform terminology for the discussion of security incidents. In an incident, an attacker executes one or more attacks to achieve specific objectives. In each attack, tools are used to exploit vulnerabilities causing a series of events that produce an unauthorized result. Each individual event is a specific action undertaken against a target. Figure 2 is the canonical representation of this taxonomy. It presents a reasonably exhaustive listing of the members of each of these classes. Figure 2 can be narrowed down to include only those issues relevant to a particular enterprise and retain only threats relevant to commercial enterprises.

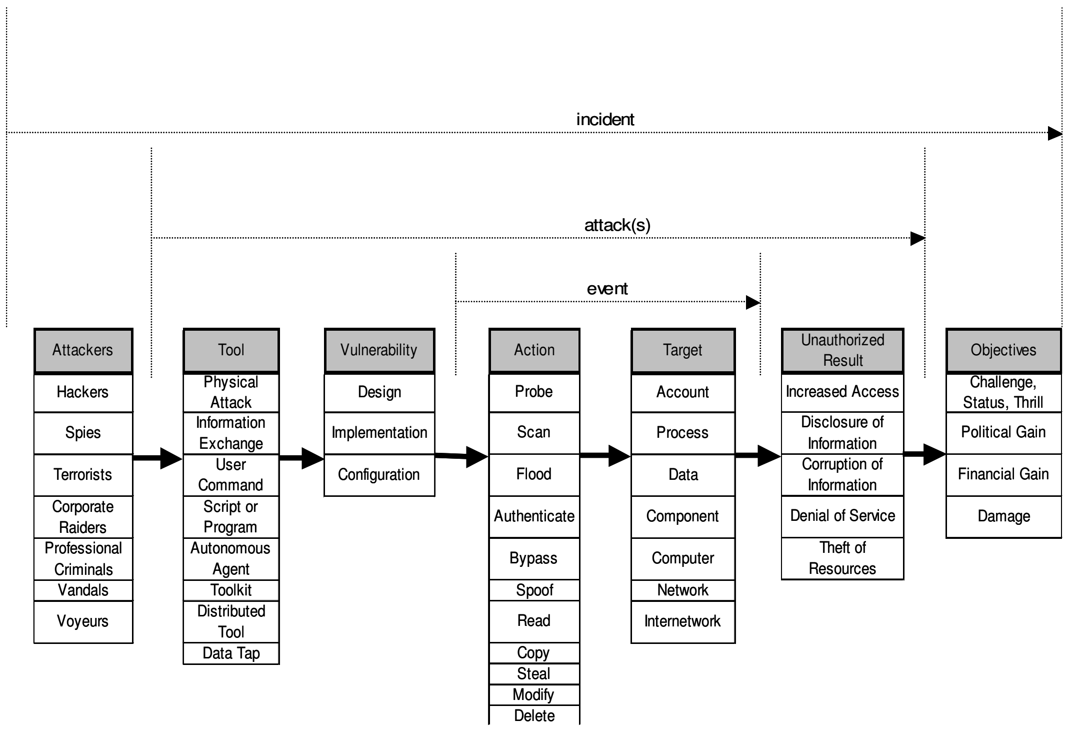


Figure 2. Computer attack incident taxonomy from [24] used in [4, 24].

The very common attacks with respect on enterprise security are discussed in detail below.

2.1.3.1 Input Exploitation attacks

Buffer Overflow: A buffer overflow occurs when an application attempts to store more data in a memory buffer than has been allocated. In languages like C/C++, the overflow overwrites adjacent memory. This can cause the application or system to crash. Carefully crafted overflows can overwrite specific locations in memory; causing arbitrary code to execute and take over the machine. In strongly typed languages like Java, memory management is handled differently and an array bound checking is part of the underlying language. Buffer overflow attacks are very unlikely to occur and much less severe for Java applications. For strongly typed languages, buffer overflows could possibly be used to create denial of service events. It may also be possible to trigger an error in the application, which may have unforeseen side effects if error handling routines are poorly implemented. The attacker can gain lots of information from error messages displayed in response to attempted buffer overflow attacks. This attack has several possible objectives, the most damaging ones are:

1. To crash the server,
2. To get control of the server, or
3. To get control of the application.

The system becomes vulnerable to such an attack when user input size is not properly validated in application code or in the software infrastructure. This attack could be detected before it occurs by finding inputs to data fields that are noticeably longer than typical inputs to the field. Vulnerability to this attack is frequently signaled in advance by a piece of software crashing unexpectedly due to segmentation faults. Several methods exist for preventing these attacks, including:

1. Proper coding practices,
2. Frequent code reviews,
3. Extensive software testing,
4. Vigilant system administration that applies the most recent patches to the operating system and software infrastructure,
5. Use of strongly typed languages like Java that are not susceptible to this attack,

6. Static code verification using tools, or
7. Implement a data input monitoring task that tracks the input lengths of data fields and signals when field length is an outlier exceeding the normally experienced field length variance.
8. Implement a code verification tool that automatically checks code written in vulnerable languages to verify that adequate data size verifications are in place.

These special safeguards will be of limited utility against buffer overflow attacks on the operating system or commercial software infrastructure. For those components, the buffer overflow attack is likely to succeed and damage the system before the data input monitoring could be executed. Use of the code verification tool would be possible only on open source components. For Java applications, it would suffice to perform penetration testing to see that unreasonably long inputs do not disturb the application.

Buffer Underflow: Buffer Underflow refers to null values given as input in the input data fields. In buffer underflow there are two distinct issues with a common cause:

1. A temporal problem caused by producer consumer patterns when the buffer becomes empty. This occurs when the producer stalls. This may be problematic for some systems. The classic example is CD/DVD burning where the laser cannot stop instantaneously.

2. A parsing problem when a data input field is empty.

The likely negative effects of this attack are:

1. For the temporal problem, a deadlock or similar problem may arise leading to a degradation or denial of service.

2. For both problems data corruption is possible.

The system becomes vulnerable to such an attack when software design or implementation does not consider the affects of null buffer lengths. This attack is signaled by the presence of empty data fields. Several methods exist for preventing these attacks, like formal protocol analysis to guard against deadlocks, livelocks and related problems. Another solution would be to perform penetration testing on the application. A PERL filter that exhaustively tests for the consequences of entering empty data fields can be used to accomplish penetration testing.

Special Character Attack: In the unexpected operator attack, carefully crafted input is sent to an application. This input has special characters in it that can cause the application to execute arbitrary code. This vulnerability can be exploited by appending the input with special characters. Prepending file names with file system navigation commands, like "../" can be used to trick the system into modifying system files. The ";" operator can be used to trick database applications into passing commands directly to an operator command shell. Wildcard operators, like "*" may result in a clever user circumventing security checks. Cross-site scripting (XSS) is one particularly virulent example, where users surreptitiously insert executable content into a data store. If done correctly, the XSS attack can result in customers executing arbitrary malicious code. Note that this vulnerability can occur whenever user supplied data is ever displayed or parsed. The likely negative effects of this attack are:

1. Execution of arbitrary code on the host,
2. Unintentional disclosure of data and violation of confidentiality,
3. Session hijacking,

4. Corruption of system files, or
5. Circumvention of security checks.

A system becomes vulnerable to such an attack when several symbols have special meanings for specific software components. Carefully crafting sequences of inputs can result in innocuous sequences of symbols being transformed into harmful instructions. This attack is signaled by the presence of specific symbols in data streams. Several prevention techniques exist, including:

1. XSS is often foiled by translating symbols into equivalent representations that are not recognized as special symbols,
2. Searching for special symbols and removing them totally from input streams, and
3. Perform extensive penetration testing. This can be achieved by creating a PERL filter that exhaustively tests for the consequences of inserting special characters into input messages.

It is worth noting that legitimate uses exist for these symbols, so that the first two solutions may either break existing applications or needlessly interfere with the implementation of legitimate applications.

2.1.3.2 Code Insertion attacks

Many modern applications are built by interfacing multiple components. Typical user actions pass through an IP stack, an XML parser, a web server, and a database engine. The request is interpreted and acted on by each of these software systems. In code insertion attacks, the attacker takes a normal system request and adds malicious instructions that speak directly to a specific software system. These attacks are quite common and potentially disastrous.

Script Code Insertion: Script code insertion is typically referred to as cross-site scripting. In the typical scenario, an application requests input from users that can be provided to other users. If a user provides content that includes scripting commands, this can result in the script executing on any host that attempts to display the web page in the future. This approach is frequently used to hijack sessions. This attack can result in:

1. Session hijacking,
2. Loss of data confidentiality, and
3. Execution of arbitrary code on other clients.

This attack can be successful when an application allows users to modify system data. Care needs to be taken to examine all data that comes from untrusted sources and disable any portion of the content that could be interpreted as scripting commands. This attack is signaled by the presence of meta-characters in user inputs. This attack can be prevented by not allowing content to be modified by untrusted parties. If the application requires this ability, user inputs need to be carefully parsed to check for executable content. This content needs to be disabled. This is done by translating meta-characters that signal the presence of script commands into equivalent encodings that are not executable.

LDAP Code Insertion: The Lightweight Directory Access Protocol (LDAP) is a simplified variant of the X.500 ITU directory standard. As a technology for retrieving hierarchically stored data, it is in some ways a competitor with SQL. As with SQL, it is vulnerable to injection attacks. If the application uses user inputs to manufacture a query, it may be possible for a malicious user to cleverly manufacture inputs that radically modify the request. This attack can result in:

1. Disclosure of data,
2. Corruption of the LDAP data, and
3. Modification of LDAP directory.

A system becomes vulnerable to such an attack when user inputs are used by an application to construct LDAP queries. This attack is typically signaled by the presence of meta-characters in user inputs used to construct LDAP queries.

This attack can be prevented by not relying on user inputs to construct LDAP queries. If the application design should require that sort of flexibility, user inputs need to be parsed carefully. In particular, the characters "(", "*", and "||" should be handled with care. If it would be possible to design the system to not allow these characters in user inputs, that would be desirable. The verification of user inputs should be done on the host, as modifications of queries through data tampering is possible at many points on the vehicle/client. It is also highly advisable to implement the LDAP directory using the principle of least privilege. All client accounts should be constrained to have minimal access rights. Alternatively, user inputs need to be carefully parsed and the presence of potentially damaging requests detected. Extensive penetration testing is required.

SQL Code Insertion: The structured query language (SQL) is the standard interface language for interactions with relational databases. Most SQL implementations also include escape characters that allow a query to include commands that are passed directly to the operating system command interpreter (shell). Once an attacker has gained access with the aid of other attacks the attacker can embed crafted SQL commands in the input requests to gain information about other accounts, personal user information, proprietary information etc. All the attacker has to do is find the parameter the application server passes to the database. This attack can result in:

1. Disclosure of data,
2. Execution of arbitrary shell commands on the server,
3. Corruption of the SQL database, and
4. Modification of the database.

The vulnerability to such an attack exists when user inputs are used by an application to construct SQL queries. This attack is typically signaled by the presence of meta-characters in user inputs used to construct queries, but other techniques exist for constructing malicious queries. This attack can be prevented by using the same techniques as in LDAP code insertion (2.1.3.4.2). Apart from these techniques escape characters including quotes, dollar signs, and semicolons should be handled with care. Since SQL is not strongly typed, the attack can also use the contents of variables to construct attacks. User inputs should not be used to construct SQL queries. Alternatively, user inputs need to be carefully parsed and the presence of potentially damaging requests detected. Extensive penetration testing is required.

2.1.3.3 Platform attacks

This subsection (2.1.3.3) cover attacks directed at the platform, which includes all non-application specific components, including operating systems, standard libraries, web servers, databases, and other off-the-shelf components. Because these components are often well-known and widely-used tools, they present special security concerns.

These tools are likely to be more secure than custom coded components due to the scrutiny that these products receive from their large user-base. However, any platform vulnerabilities that are discovered are published, and in most cases, a software revision (i.e. patch) is made available soon after. If patches are not promptly installed, the system becomes vulnerable to well-known attacks. Unfortunately, new patches also introduce new functionality and security bugs, and some level of testing or redundancy is advised.

We first cover the pre-attack phase of probing and scanning, which is used to identify information about the system and the various platforms that it uses. These are not attacks but methods by which an attacker can gain information about the system.

Port Scanning and Probing - Port scanning is used to find open network ports, which correspond to server applications that support network access. There are various port scanning utilities that are readily available (e.g. nmap, ping, trace analyzers).Once the open ports are determined, an attacker can send various requests probing for information (e.g. the version of the web server). Probing refers to examining a host and testing its responses. In Probing attacks an attacker scans a network of computers to gather information or find known vulnerabilities. A probe attack involves an automated program that scans a host's ports, looking for running services that respond to queries. These probes can be anything like invalid credentials, invalid requests, junk packets. The attacker then attempts to exploit weaknesses in responding services. The server would send error messages in response to these invalid requests. Though the attacker does not gain access to the server these error messages can equip the attacker with vital information. This is the very first thing an attacker will do when he is about to attack. A successful probing attack will inform the hacker about vulnerabilities in the network which he can use to stage hazardous attacks like DoS, Buffer Overflow etc. A probing attack generally does not harm a system directly; it just leads to compromise of valuable information that can be misused later.

Tracing/Analyzing and Eavesdropping - There are various trace analyzers that can trace and analyze connections. Though the payload is encrypted, other information like the IP address and port number are not. Analyzing such traces can also provide information to the attacker. Like probing and scanning, trace analysis can also provide the attacker with information like product type and versions. The attacker can then search for known vulnerabilities and try to attack the server. Unlike probing and scanning, tracing can be done passively (i.e. eavesdropping), so that it is undetectable.

The objective of such an attack would be to gain platform information about the system, such as product versions and configurations, for the purpose of finding attack points and vulnerabilities. Any information that can be extracted from invalid requests or unencrypted data presents a vulnerability for collecting data, including addresses, port numbers, response headers, protocols, and error messages. To prevent such attacks some of the information sources can be removed (e.g. unnecessary open ports), but not all useful information can be removed (e.g. the destination address can not be hidden, and hiding the web server version can be difficult). Error messages with information about the system should not be sent back to the host. The user and the client application do not need to know about the details of system errors, and so only user level descriptions should be sent over the connection. Also messages sent must be encrypted. This prevents passive eavesdropping. Client side eavesdropping (after encryption) is prevented through tamper resistant packaging.

This attack should be addressed through encryption and client-side security measures. System security should not rely on the secrecy of information that can be acquired through eavesdropping.

This part of the subsection (2.1.3.3) continues coverage of attacks directed at the platform-level, including all non-application specific components. The server components can be attacked using various forms of input exploitation. More specifically, platform attacks are very likely because attacks on these products are widely known and available. The Web server, Application server, database and Identity store can be attacked by using known vulnerabilities like buffer overflow, misconfiguration (access rights), and other known exploits.

Unknown Exploit: This attack considers the special case of buffer overflow in the platform components. Buffer overflows of platform components can be further divided into known exploits and new exploits. Known exploits will be covered in the next subsection and the current attack will focus on new exploits. While known buffer overflow have a better chance of being detected, new buffer overflow exploits will be difficult to detect. As with any buffer overflow attack, this attack has several possible objectives, the most damaging ones are:

1. To crash the server,
2. To get control of the server, or
3. To get control of the application.

By gaining control of a platform component, access control and any application security measures can be circumvented by the attacker. The system becomes vulnerable to such an attack when user input size not properly validated in application code or in the software infrastructure. Unlike application specific code, buffer overflows can not be prevented at the source, because the source code for platform components are not under the control of an enterprise. Platform buffer overflows must be prevented using external input validation and patching – in cases where the supplier has removed a buffer overflow vulnerability. Since this attack is focusing on unknown exploits, input anomaly detection or post-attack-detection are the only options. This attack could be detected before it occurs by finding inputs to data fields that are noticeably longer than typical inputs to the field. Vulnerability to this attack is frequently signaled in advance by a piece of software crashing unexpectedly due to segmentation faults. Another solution would be to filter unexpected packets through stateful monitoring.

Known Exploit: This attack considers known attacks on platform components that have been published and either patched or otherwise handled by commercial tools. Known exploits can have several possible objectives, the most damaging ones are:

1. To crash the server,
2. To get control of the server, or
3. To get control of the application.

By gaining control of a platform component, access controls and any application security measures can be circumvented by the attacker. A system becomes vulnerable to such an attack if user input size not properly validated in application code or in the software infrastructure. Known exploits can be detected using signature-based identification in IDS/antivirus/firewall products. Known exploits can be prevented using signature-based identification or patching by the product provider. Also an Intrusion Prevention System (IPS) can be deployed to detect and drop known exploits (prior to patch release). Signatures and patches must be kept up to date.

Misconfiguration: Just as harmful as a known exploit, misconfiguring a component can render the entire system defenseless. A trivial example would be leaving the default administrator account and password set. Misconfiguration can be difficult to detect through monitoring, but many commercial tools will actively check for common misconfigurations. There are also configuration guides for specific products that provide either a secure configuration or a checklist for creating one.

Abuse of Error Logs and Audit data: This final platform-level category covers a single attack, abuse of data from error logs and audit logs. An Attacker can gain information on user accounts and system details from the error log data and the audit data. The error log data of the application server has a log of all the successful authentications and failures. Gaining access to this log can give away vital information of the system and personal user information. This needs to be very well protected. Similarly the audit data of the database also has vital information that needs to be protected. Unauthorized access to these logs and audit data can be gained by changing permissions (broken access control), Input exploitation etc. The attacker can be on the company intranet with a few rights. This attacker can change or elevate his permissions in order to view the logs or the audit data. Additionally, an attacker can modify audit logs and error logs to hide his or her attack or actions. The objective of such an attack would be to gain access to logs and audit data for the purpose of reading the data or modifying the data. A system will be vulnerable to such an attack if file permissions are not set correctly. Also if a system is vulnerable user input exploitation, an attacker can use that vulnerability to get access to error logs and audit logs. This attack can be detected by checking the log files for unauthorized user access. Access to the logs and audit data can be prevented by strict input validation, restricted file permissions and mandatory access controls. For Linux platform, SE-Linux security module should be used with mandatory access control to prevent unauthorized access. To prevent modification, a duplicate real-time copy of the log should be maintained (e.g. using a virtual printer).

2.1.3.4 Session ID Attacks

Session ID attacks allow intruders to access a system by circumventing the authentication process. The attacks discussed here are application level attacks. Similar problems exist at the network and host levels [30]. Network layer session hijacking attacks include the man-in-the-middle attack (where attackers insert themselves as transducers between client and server), and packet injection attacks (where spoofed packets replace legitimate packets in a TCP session). Host level attacks are possible by using port forwarding, for example. It is also possible to subvert the network DNS directory system, so that attempts to access a server are redirected. To avoid these attacks, it is essential that strong cryptographically sound credentials be used in the authentication process.

Brute force: Session ID's are used to identify legitimate sessions. For web applications, since the HTTP protocol is stateless, a "magic cookie" is often used to store the session ID. The value stored in this cookie uniquely identifies the user's session to the web server. This allows a user to log in to the service and their authentication credentials are associated with the magic cookie on the web server.

If the session ID is not generated randomly from a sufficiently large range of values, it becomes possible for an attacker to predict future values of the "magic cookie" with a non-trivial likelihood of success. This means that an attacker can calculate a session ID using brute force, attach to the server, and quite possibly be able to use the system by masquerading as a valid user with an active session. This is done by an attacker that has never had any contact (direct or indirect) with the poor victim, whose account is probably being charge for the intruder's actions.

Copy Session ID: In the copy session ID attack, the intruder manages to steal the session ID from a valid active session. One way to steal the session ID is by staging a successful cross-site-scripting (XSS) attack. The script finds a channel for transmitting the session ID to the attacker. This attack can result in session hijacking which allows the intruder to circumvent the authentication process. This attack is successful when the intruder succeeds in accessing the session ID credentials of a legitimate user. For most web applications, no additional credentials checking are performed. The session ID can be compromised through sniffing. Since the intruder possesses legitimate credentials, that data item is not a reasonable signature for attack detection. Should the user sessions follow a typical sequence of activities, such as accessing a tree structure of options menus, it should be possible to detect sequences of commands that either violate the system structure, or are highly unlikely. This concept directly contradicts the stateless nature of IP, which would make use of this approach very problematic. Keeping session ID credentials private can prevent this attack. Alternatively, additional authentication checking could be performed. For example, using the current IP address of the client as an additional credential could be effective. Also, formal analysis of the protocol used to set up session IDs can be done. Penetration testing of the system to see how feasible it is to find session IDs should be done.

2.1.4 Attack Trees

Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes [31]. Figure 3 shows an example of an attack tree to attack the database. The root of the tree is the goal of the attacker. The root has sub trees that have leaves and so on. The last leaf nodes denotes the information that the attacker starts with to achieve is final goal i.e. root. Simple arrows represent the 'OR' condition i.e. alternatives. If the arrows are grouped together, it means an 'AND' i.e. the attacker requires all the conditions that are grouped to proceed.

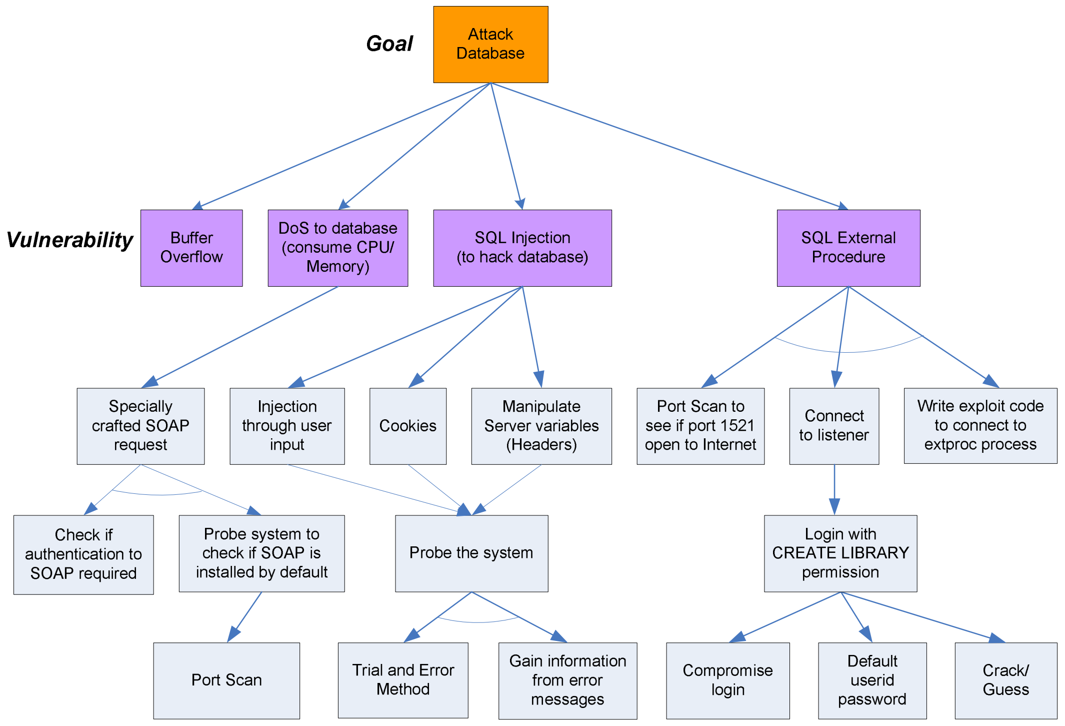

Goal
Attack Database
Vulnerability
Buffer Overflow
DoS to database (consume CPU/ Memory)
SQL Injection (to hack database)
SQL External Procedure
Specially crafted SOAP request
Injection through user input
Cookies
Manipulate Server variables (Headers)
Port Scan to see if port 1521 open to Internet
Connect to listener
Write exploit code to connect to extproc process
Check if authentication to SOAP required
Probe system to check if SOAP is installed by default
Probe the system
Login with CREATE LIBRARY permission
Port Scan
Trial and Error Method
Gain information from error messages
Compromise login
Default userid password
Crack/ Guess

Figure 3. Attack tree to attack the Database.

Security auditors (red teams) commonly use attack trees to analyze infrastructure security. They start eliminating the last leaf nodes by using preventive techniques. For e.g. to eliminate a node that says Port Scanning, an auditor would have to see if there any open ports and then close any unused open ports. If the auditors are able to cut off all these last leaf nodes, they can conclude that the attacker will never be able to reach the root of the tree. Figure 4 is another example of an attack tree to attack the Web Server/Database using the Buffer Overflow vulnerability.

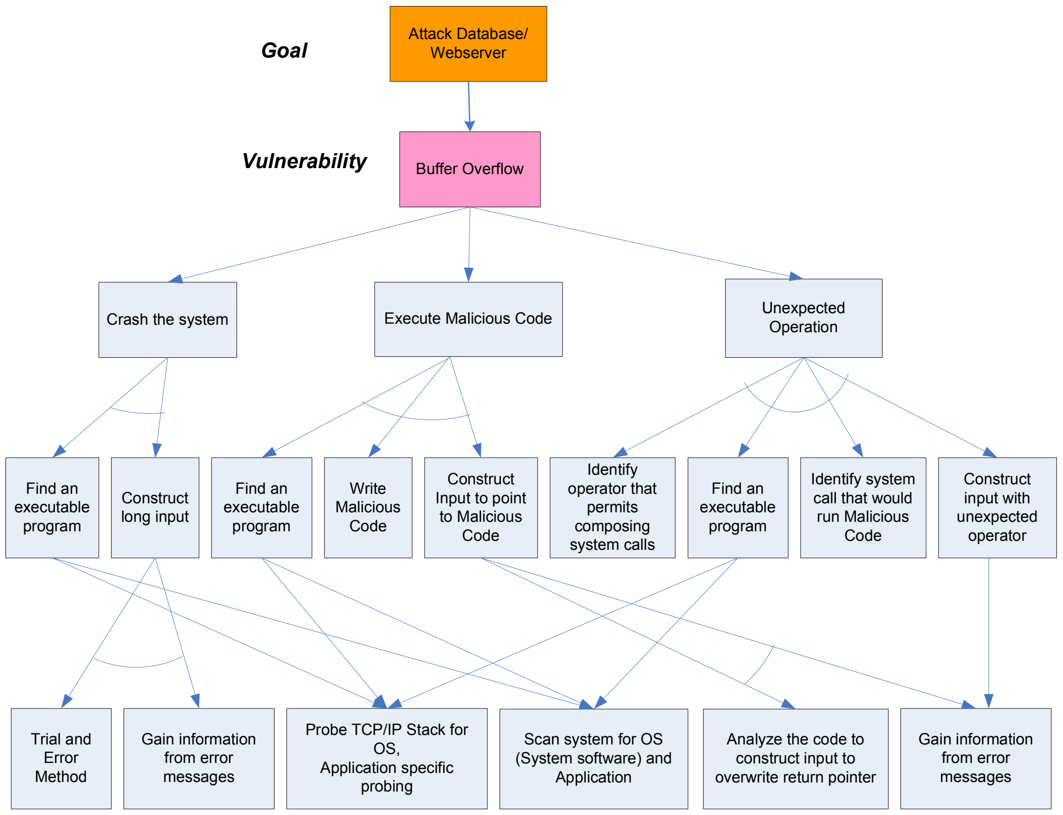

Goal
Attack Database/ Webserver
Vulnerability
Buffer Overflow
Crash the system
Execute Malicious Code
Unexpected Operation
Find an executable program
Construct long input
Find an executable program
Write Malicious Code
Construct Input to point to Malicious Code
Identify operator that permits composing system calls
Find an executable program
Identify system call that would run Malicious Code
Construct input with unexpected operator
Trial and Error Method
Gain information from error messages
Probe TCP/IP Stack for OS, Application specific probing
Scan system for OS (System software) and Application
Analyze the code to construct input to overwrite return pointer
Gain information from error messages

Figure 4. Attack tree to attack the Web Server / Database using the Buffer Overflow vulnerability.

CHAPTER 3

BACKGROUND ON DENIAL OF SERVICE ATTACKS AND EXISTING MITIGATION TECHNIQUES

3.1 Denial-of-Service and Distributed Denial-of-Service Attacks

A DoS attack is an explicit attempt by attackers to prevent legitimate users of a service from using that service [10]. DoS attacks are perpetrated either locally or over the network. On a multi-user system, any user consuming an inordinate amount of resources launches a simple DoS attack on other users. This is stopped by attentive system maintenance. Sabotaging equipment is another simple DoS attack; countered by physically protecting devices. Remote DoS attacks using the Internet are of primary interest.

DoS attacks can be classified as system exploit attacks and network attacks [4]. System exploit attacks exploit security vulnerabilities specific to the system. These can cause a server or service to crash or significantly reduce performance. Example of this attack would be the famous Ping of death exploit where a ping packet (echo request datagram) with more than 65,507 octets of data is sent to a machine. This malformed ping packet results in the crashing of a computer [32].

Networking attacks take advantage of vagueness or weakness in protocol specifications. Known Internet DoS exploits include: attacks on root Internet name servers, attacks on merchant and media sites, and the Slammer worm [4]. The network traffic generated by Slammer triggered DoS events for other systems.

Most networking attacks flood the network with spurious traffic to block legitimate traffic. In the case of SYN floods, the attacker fills the buffer for the TCP connections by sending multiple connection requests. In some earlier operating system version's the size of the buffer was too small and could easily be filled [33]. More brute force attacks are also possible, where a large volume of packets fills all of the available bandwidth. Packet queues fill on intermediate machines and legitimate packets get dropped. Packet dropping forces TCP to retransmit packets while flow control slows the packet throughput rate. The network is then unusable for its intended purpose.

A Distributed Denial of Service (DDoS) attack is a networked DoS where nodes work together. A typical DDoS attack contains three stages as shown in Figure 1. It follows a hierarchical model, with one or more attackers controlling a handler, which in turn controls the hordes of zombies that execute the commands relayed to them [34].

The first stage is the communication between the attacker and his handler. In the second stage the attacker gains access to multiple machines over time and plants zombie processes i.e. installs attack tools in these compromised systems via the handlers [35]. This way the computers are turned into zombies. In the third stage, the attacker sends an attack command to the handlers which in turn relay this message to the zombies through a secure channel to launch an attack against the victim [34]. The communication between the attacker and the handler, and between the handler and the agents is referred to as the control traffic of the network, whereas the communication between the agents and the victims is referred to as the flood traffic.

A zombie is a daemon that performs the actual attack [36]. On receiving the attack command the zombies, located on different computers, launch the attack on the target. The random nodes send replies to the target. Hundreds or thousands of zombies working together swamp the bandwidth of the target's network connections. In the recent DDoS attack on Estonia, reports claim that there were more that about 1 million 'zombies' were compromised to launch the DDoS attack [1]. A DDoS attack is difficult (or almost impossible) to avoid. The attacker does not need to access any machine on the same sub network as the target. It is currently impossible to enforce stringent security on all the Internet Autonomous Systems (AS's).

3.2 Related Work in Mitigation of Distributed Denial of Service

The complex nature of the DDoS attacks makes it very difficult to prevent. Most of the culprits involved in the DDoS attacks are never caught as the attackers can spoof their identity in order to make it harder for the target of the source to identify the root of the attack [13]. In order to remove himself from view, the attacker introduces additional layers between the victim host(s) and him/herself [34]. Also there is a lack of detailed information of the attack. A lot of time and money is spent on the analysis of the DDoS attacks. By the time the analysis is over, the attackers have found other ways of executing DDoS attacks.

The seriousness of the DDoS problem and increase in the frequency over the past few years have led to the proposal of numerous defense mechanisms. In [3] the authors classify defense techniques as either preventive or reactive. Preventive techniques identify loopholes in current systems or protocols and correct them. Preventive techniques cannot guarantee that systems and protocols are 100% secure, but they can reduce the prevalence and consequences of DDoS attacks.

Reactive techniques try to detect attacks quickly and respond. It is important that reactive technologies have low false negative rates. Reacting to falsely perceived DDoS attacks can have consequences just as bad as the DDoS attacks themselves.

In [37] the authors classify the defense techniques in another way as either fair resource allocation or filtering / rate-limiting techniques.

Fair resource allocation that is preventive in nature quantifies the resources available at the server and distributes them fairly among all the clients. Resource allocation can use Quality of Service (QoS) techniques like Weighted Fair Queuing (WFQ), class-based WFQ etc [38], [8]. A related approach [39] views DDoS attacks as a resource management problem. This assumes that servers are close to the backbone network with high capacity pipes and see the full force of the attack traffic that has gone through aggregation inside the network. So before the aggressive packets can converge to attack or disturb a server the routers along the forwarding path regulates the traffic, thus avoiding an impending attack.

Preventive techniques prevent high-rate DDoS attacks but sometimes also affect legitimate traffic. It is very difficult to distinguish a DDoS attack from a flash event [2], [40]. A flash event occurs when large amounts of expected or unexpected traffic from legitimate clients suddenly arrive at a system. This is also called the "Slash-Dot Effect," For example; a new car model website might have increased traffic on the day it is announced. Preventive techniques affect legitimate traffic in two ways,

1. Since these techniques cannot distinguish between attack and legitimate traffic, legitimate traffic will be throttled along with attack traffic.
2. Since these techniques are unable to detect flash events, they might have to excessive false positive rates.

Reactive filtering and rate-limiting defenses are attack-specific. They first identify attack traffic and then use filters to remove the attack traffic [15]. There are only two ways to identify the source of the attack packet, the source IP address or IP header values [41], [42], [43], [44], [45], [46]. Unfortunately, the way the IP protocol is implemented these values can be easily spoofed. To prevent IP spoofing, some administrators use ingress and egress filtering [47]. Since ingress and egress filtering is not used by all the end point routers, its use is not very effective. Attacker spoofing of IP addresses on the same subnet can also negate ingress and egress filtering. Packet filtering is typically ineffective if attack traffic cannot be identified.

There are other methods like [48] where the destination must give permission to the sender before packets are sent. These techniques have excessive overhead, cause unnecessary delays, and still can not prevent IP spoofing.

Some detection techniques [49], [50] use trace-back to find the routers and links used in attacks. The origin of a flood attack can be found, but the attack will not be stopped immediately. These trace-back approaches also require excessive overhead. Trace-back is effective for detecting a single attack source, but will be ineffective for DDoS attacks. As the number of attack sources increase, trace-back becomes increasingly difficult [49].

Surveys [2], [40] and [51] classify existing DoS defense techniques. [2], [40] conclude that none of the techniques address real-world concerns. The authors doubt the robustness of DDoS detection approaches and suggest relying on network administrators to manually detect DDoS attacks.

The approach we present builds robust networks that tolerate DDoS attacks. On a given network, we determine the resources an attacker needs to stage a DDoS attack. This gives direct insight into the network vulnerabilities that make DDoS attacks possible. This information can be used in turn to make the networks less susceptible to attack. In a similar approach [52] Blue tries to restrict the attacker's network use by solving the bi-objective maximum-flow network-interdiction (BMXFI) using Lagrangian relaxation to reduce the time complexity of the problem. In this thesis we use concepts of Combinatorial Game Theory, given in Section 6.1.1, to solve the problem.

CHAPTER 4

DISTRIBUTED DENIAL OF SERVICE MODEL

4.1 Problem Statement for DDoS Games

The idea presented here is based on the work in [4] that finds the complexity of optimal DDoS attack design for a given graph and distributed application. In this work we describe a two player game played on a physical graph. A computer network is modeled by a directed physical graph structure in which computers are graph nodes and links connecting computers are graph arcs.

4.1.1 Physical Environment

The Physical environment (computer network) is represented by a directed graph structure (*EG*). This computer network consists of *N* nodes. As an illustrative example, we will discuss a model of a real computer with about 500 nodes connected by about 1500 arcs. *EG* is represented by:

$$EG = [EV, EE] \tag{4.1}$$

where *EV* is the set of vertices or nodes (*computers*) and *EE* a set of directed edges or links.

Each element of *EV* has an associated capacity value representing its processing power. This is represented as a vector.

Each element of *EE* has an associated capacity value representing the link's communications bandwidth available. *EE*'s connectivity matrix describes connections between network nodes. *EE*'s capacities are represented by a square connectivity matrix of the order ($N \times N$) where each element is either the capacity over the edge or is a 0 if there is no link connecting two nodes. Nodes do not communicate with themselves over the network i.e. in the connectivity matrix the edges (1,1), (2,2), (3,3)…….(N, N) will always be zero (the diagonal of *EG*'s connectivity matrix will always be zero). The local communications bandwidth on each node is considered infinite.

4.1.2 Blue (Virtual) Environment

The Virtual (Blue) environment is a distributed application comprising of the distributed programs. For the successful execution of this distributed application within the network, the distributed programs placed on the physical nodes of *EG* must be able to communicate with each other at all times. This environment is represented by a "logical" directed graph structure (*BG*). One or more programs can run on the same computer. The Blue Environment consists of *M* blue nodes. *BG* is represented by:

$$BG = [BV, BE] \tag{4.2}$$

where *BV* is a set of vertices or nodes (distributed programs) and *BE* is a set of edges or links.

The capacity for each element of *BV* is the amount of CPU load it can provide. *BV*'s capacities are given as a vector.

The capacity for *BE* represents communication requirements. Connectivity matrix *BE* describes connections between nodes in the network. *BE*'s capacities are represented by a square ($M \times M$) connectivity matrix where each element is either the capacity required over the edge or 0 if no link connects two nodes. Nodes do not communicate with themselves over the network i.e. in the connectivity matrix the edges (1,1), (2,2), (3,3)…….(M, M) are always zero (the diagonal of *BG*'s connectivity matrix is always zero). The local bandwidth on each node is considered infinite.

The two players are:

1. *Player 1:* Player 1 is a distributed application on the network denoted by the color *Blue*. A set of programs consume CPU resources on "physical" nodes. For each pair of programs, there is a known communications bandwidth requirement. These constraints define a "logical" graph. The set of "feasible configurations" is the set of mappings of logical nodes to physical nodes, where the logical graph's CPU and communications needs are satisfied by the physical graph.

2. *Player 2:* Player 2 is an attacker that is denoted by the color *Red*. Red places zombie programs on the physical nodes. These processes can send network traffic over the physical edges to consume network resources. If the Red zombies consume enough communications bandwidth to make the physical graph unable to satisfy one of the logical graph's constraints, Blue's configuration is disabled. His aim is to disrupt the functioning of blue in two ways,
    a. *Exhaust Node Capacities:* Disrupt the ability of a distributed program placed on the physical node of *EG* by exhausting CPU load and
    b. *Exhaust Arc Capacities:* Disrupt the ability of a distributed program to communicate with another distributed program

The aim of our work is to find the connectivity bottlenecks and vulnerable nodes that are absolutely necessary for the Blue network to stay connected. Further we find the minimum number of zombies and the minimal amount of flow required to attack bottleneck links and vulnerable nodes. By analyzing the strategies of the two players, conclusions are drawn about the conditions necessary to successfully stage a DDoS attack. Our results show a strong relationship between the connectivity of the graph and the ability of individual network members to resist DDoS attacks. The results obtained can be used to design robust networks.

4.2 Key Concepts

Graph Theory: In this section, some of the concepts of graph theory [5] used are explained.

Graph: A graph is a graphical representation of a network, where the nodes of the graph are hosts or computers and the arcs of the graphs are the links connecting the computers.

Directed Graphs: A directed graph is a network whose elements are ordered pairs of nodes. If two nodes *A* and *B* are connected by a single directed arc ($A \rightarrow B$), this does not mean that node *B* is also connected to node *A*. Flow can be sent from Node $A \rightarrow B$ but not visa versa. These graphs have nodes and arcs with associated numerical values (like costs, capacities, etc).

Connectivity: Two nodes *A* and *B* are connected if the graph contains at least one path from node *A* to node *B* [5]. A graph is connected if every pair of its nodes is connected.

Adjacency Matrix: This matrix stores the network in the form of a '$N \times N$' matrix that gives the connectivity between all the 'N' nodes of the network. The matrix has a row and column corresponding to every node. If the arc between two nodes *A* and *B* exists (*A*

→ *B*), the value or capacity or simply 1 is written in the A^{th} row and B^{th} column else a 0 is written.

Source Node: The node that is the starting point for a flow is called a source node.

Sink Node: The node in which a flow terminates is the sink node.

A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink, which has more incoming flow. For more information on graph theory see [5].

Max-Flow: In a network graph, the max-flow is the maximum possible flow that one can route from one node (source) to another (sink) [5]. For the physical network *EG* we calculate the max-flow from *N* sources to *N* sinks. The max-flow from node *j* to node *k*, determines the logical link capacity available to Blue nodes (programs) placed on physical nodes (computers) *j* and *k*. Many techniques can be used to find the max-flow, we use the Highest Label Preflow Push Algorithm [5]. It has the lowest running time in practice - $O(n^2m)$ [5]. Consider our example application which requires calculating the max-flow from 10,000 sources to 10,000 sinks.

Min-cut: The min-cut is the smallest set of edges or arcs that is absolutely necessary for a source to communicate to a sink. The removal of these edges from the network graph completely disconnects the source node from a sink node. Blue must safeguard these arcs to maintain connectivity over the network.

To illustrate the concepts of max-flow and min-cut, we find let us calculate the max-flow and min-cut from source node 1 to sink node 6 in Figure 5. There are two paths from node 1 to the node 6. The first path is (1-2-4-6) and the second path is (1-3-5-6).

The maximum flow over (1-2-4-6) is bounded by arc (1-2) which has capacity 2. The second path (1-3-5-6) is bounded by arc (5-6) with capacity 2. Since each of the only two paths from 1 to 6 has capacity two, and the two paths are disjoint, the max-flow from 1-6 is their sum (4). The min-cut is the smallest number of edges with minimum capacity whose removal will disconnect the source from the sink. In Figure 5, arc (1-2) is the minimum capacity arc in path (1-2-4-6), and arc (5-6) is the minimum capacity arc in path (1-3-5-6). Removing these two arcs disconnects node 1 from node 6. So the min-cut is the set of arcs 1-2 and 5-6.

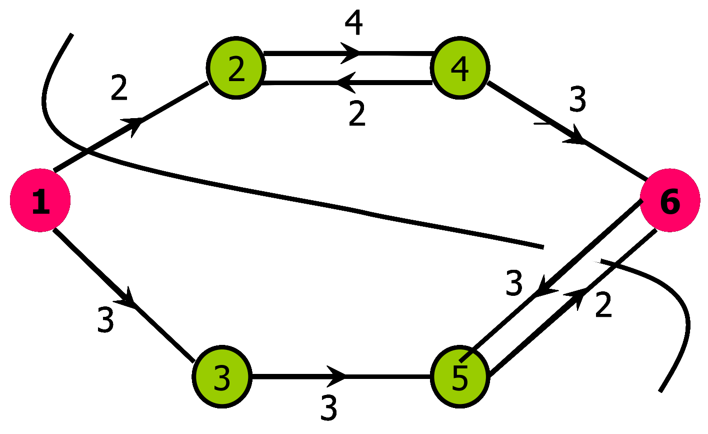


Figure 5. Max-flow and min-cut for a directed graph structure.

Note that "*The maximal amount of flow is equal to the capacity of a minimal cut",* which is the max-flow min-cut theorem [5].

CHAPTER 5

DISTRIBUTED DENIAL OF SERVICE MITIGATION APPROACH – TRAFFIC FLOW

In this section we first define the goal of each player followed by the strategies each player uses to either win the game or maximize their expected payoff.

5.1 Player 1 - Blue

Player 1 (Blue) is a distributed application on the network. This distributed application consists of programs executing on physical nodes. The distributed programs consume CPU resources on the local physical node. Each pair of programs has a known communications bandwidth requirement. These programs must communicate with each other in order to execute successfully. We first determine the possible positions (computers) where Blue programs can reside. This gives us a set of "feasible Blue configurations" that is the set of mappings of logical nodes to physical nodes, where the logical graph's CPU and communications needs are satisfied by the physical graph.

Aim of Player Blue:

1. To ensure that the Blue - distributed programs remain connected at all times.
2. If a particular Blue configuration is attacked, Blue switches to another available Blue configuration from the set of Blue configurations to a configuration that Red cannot attack or is less affected by the Red attack.
3. Blue tries to find a "loopy" [7] game where it can always return to a previous configuration. A detailed explanation on loopy games is given in section 6.1.1.

Strategy: Blue first finds the set of possible Blue configurations. Blue can then reconfigure by moving to another Blue configuration once attacked. To find the set of feasible configurations for Blue i.e. the set of mappings of *BV* (distributed program – logical graph) onto *EV* (physical graph) Blue has to satisfy two classes of constraints:

1. Node Capacity Constraints
2. Edge Capacity Constraints

5.1.1 Node Capacity Constraints

Statement: The sum of the CPU requirements for the set of nodes from *BV* assigned to each element of *EV* is less than or equal to the CPU bandwidth of that element.

A blue node can be placed on a physical node whose node capacity (processing power) is greater than or equal to the node capacity (CPU load) of the blue node. More than one blue node can be placed on the same physical node provided that the sum of all the nodal capacities of the blue nodes placed on the single physical node are less than or equal to the capacity of the single physical nodes. This can be represented as:

$$\textit{Nodal Capacity of a Physical Node} \geq \textit{Nodal Capacity of Blue} \qquad (5.1)$$

For e.g. consider an example shown in Figure 6 where there are six physical nodes with a nodal capacity, *EV* = [5, 2, 1, 3, 1, 2] and two blue nodes with a nodal capacity, *BV* = [3, 1].

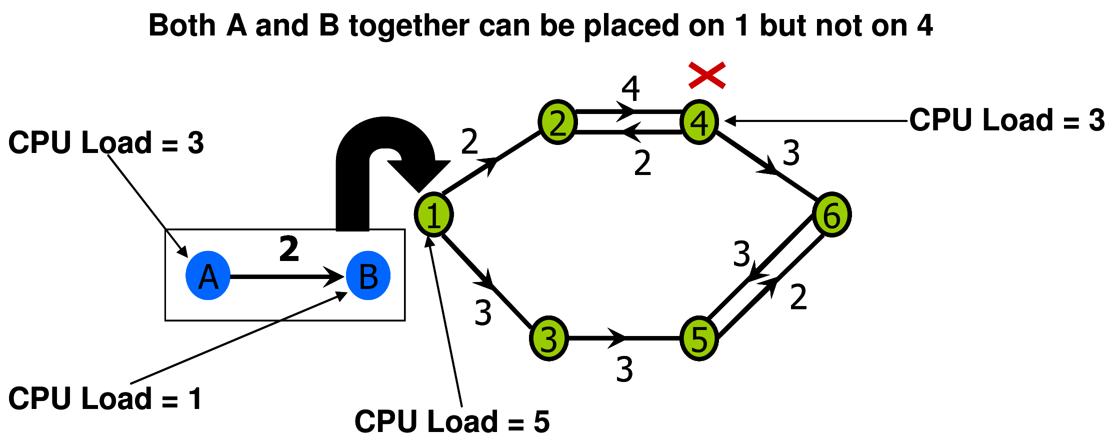


Figure 6. Node Capacity Constraints.

Blue node *A* can be placed individually on the physical nodes 1, 4 and 5 and Blue node *B* can be individually placed on any physical node. Blue nodes *A* and *B* together can be placed only on node 1, as CPU capacity requirement of *A* and *B* together is 3 + 1 = 4 and this value is less than the CPU requirement of available CPU capacity at physical node 1 which is 5. The available communication capacities between these two Blue nodes placed on 1 becomes infinite as they are on the same node and do not require any communication bandwidth to remain connected. In the same figure, Blue nodes *A* and *B* together cannot be placed on physical node 4 as the available capacity requirement of physical node 4 which is 3. Though node *A* and node *B* individually can be placed on 4, since the CPU capacity of node *A* is 3 and node *B* is 1 which are both less than the available CPU capacity of physical node 4. Once we have satisfied the nodal capacity constraints we further filter out the possibilities by checking the arc/edge capacity constraints.

5.1.2 Edge Capacity Constraints

*Statemen*t: For each element be_{ij} of BE connecting two elements of BV (bv_i and bv_j), where bv_i (bv_j) is mapped to pv_i (pv_j) the max-flow [5] on EG from pv_i to pv_j must be greater than equal to the bandwidth requirement of bv_{ij}. If pv_i and pv_j are on the same node, the value of the max-flow is infinite.

Two blue nodes can be placed on two different physical nodes if the arc/edge capacity (computational and communication requirement) between the two blue nodes is less than or equal to the arc capacity (bandwidth) of the two physical nodes. So this check can be represented by:

$$\textit{Max - Flow between two Physical Nodes} \geq \textit{Arc Capacity of two Blue Nodes} \quad (5.2)$$

If two blue nodes are placed on the same physical node i.e. they satisfy the nodal capacity constraints then we do not have to check for the arc capacity constraint as two blue nodes on the same physical node have infinite communication bandwidth available. This is illustrated in Figure 6.

To satisfy the communication requirements between two Blue nodes placed on different physical nodes we need to determine the maximum bandwidth available between two physical nodes. To determine this maximum available bandwidth between any two nodes we calculate the max-flow [5] which gives us the maximum available bandwidth. A blue node may satisfy the constraints of many physical nodes. So we will have a number of possibilities. We will thus get a set of lists of possible Blue positions on the physical nodes. We denote them as Blue configurations.

So far, two Blue nodes may be placed on physical nodes as long as the arc capacity constraints are satisfied. But we still need to cross check that the capacity constraints are satisfied simultaneously for all Blue nodes. We need to check that all outgoing arcs from any given Blue node and all incoming arcs to that same Blue node satisfy the capacity constraints simultaneously without over committing available bandwidth. To incorporate this, three verifications were carried out on the lists of Blue configurations.

5.1.3 Verification 1

Verification 1: In this verification we check if all the outgoing arcs from a particular Blue node satisfy the capacity constraints at once. This is done by carrying out a row wise connectivity check for all the blue nodes. Consider the example in Figure 7.

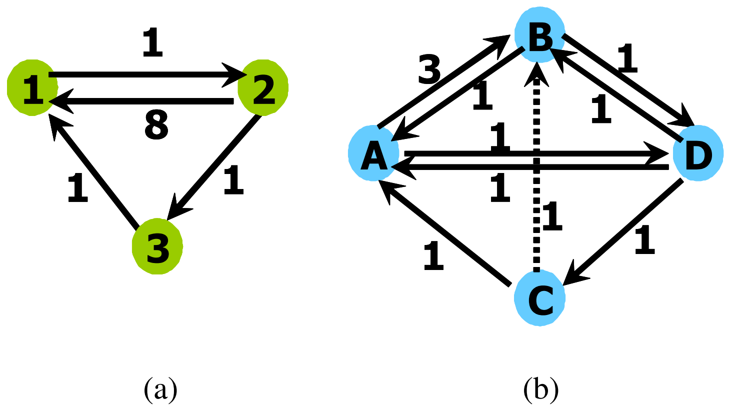


Figure 7. Connectivity Graphs (a) Physical Connectivity Graph (b) Blue Connectivity Graph.

The node capacity vectors for this example are represented by:

$$EV = \begin{bmatrix} 1 & 1 & 5 \end{bmatrix}, \qquad BV = \begin{bmatrix} 3 & 2 & 1 & 1 \end{bmatrix} \tag{5.3}$$

The connectivity matrices of the physical and blue nodes respectively are given by:

$$EE = \begin{bmatrix} 0 & 1 & 0 \\ 8 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad BE = \begin{bmatrix} 0 & 3 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{5.4}$$

The first Blue node considered is node *A*. It requires a capacity of 3 to talk to node *B* and a capacity of 1 to talk to node *D*. When a particular arc satisfies the node and arc capacity constraint, the first verification is carried out. A Blue node *A* can be placed on a particular physical node if *A* can be placed on that physical node for both the arcs *A* is connected to i.e. (*A*-*B*) and (*A*-D). In our case for arcs (*A*-*B*) and (*A*-D), *A* can be placed on physical node 3. But when we do the row-wise check for node *D* we deduce that for arcs (*D*-*A*), (*D*-*B*) and (*D*-*C*) node *D* can be placed on physical nodes 2 and 3 but not on 1. The same row-wise check is performed for all the Blue nodes.

5.1.4 Verification 2

Verification 2: Verification 2 is similar to Verification 1 except that instead of the outgoing arcs we check for all the incoming arcs to a particular Blue node. This is done by carrying out a column-wise connectivity check for all the Blue nodes. In (5.4) we now look at the columns for all the Blue nodes. For example, Blue node *A* is connected to *B*, *C* and *D* with a capacity requirement of 1 each. Blue node *A* can be placed on a particular physical node if *A* can be placed on that physical node for all the three arcs (*A*-*B*), (*A*-*C*) and (*A*-*D*). A similar check is performed for all the blue nodes.

5.1.5 Verification 3

Verification 3: This verification checks if placing combinations of blue nodes on the physical nodes leads to exhausting the respective arc capacities between the physical nodes. We find out if for any blue arc the amount of free capacity on the physical arcs becomes negative. For this verification we need to calculate the min-cut for all the Blue arcs: the min-cut gives the bottleneck edges that are absolutely necessary for a source to communicate to a sink. To do this we perform the following check for all the arcs. Say we have a Blue arc (*A*-*B*) placed on physical arc (1-2). The available excess capacity e is given by:

$$e = EE(1,2) - (I \times G) \qquad (5.5)$$

where,

$$I = BE(A,B) \div Maxflow\ from\ arc\ (1-2) \qquad (5.6)$$

and if '*e*' is negative we discard the respective arc. *EE* is the Connectivity matrix of the physical graph, *BE* is the Blue connectivity matrix and *G* is the min-cut value of arc (1-

2). One might argue that the last verification does what verification 1 and 2 do separately. The reason for this is that there are too many possibilities and it might take forever to calculate the Blue node positions as the number of physical nodes could increase to thousands of nodes. The calculation time is minimized by removing the arcs that are least likely to meet the constraints.

5.1.6 Feasible Blue configurations

Final Blue configurations: The results obtained from the three verifications are used to give the Blue nodes their physical locations. The various combinations of these physical nodes are then checked for consistency with the Blue nodes by verifying if they form a cycle. After forming the Blue configuration, each Blue configuration is cross checked to verify if all the communication capacities available on the physical arcs are sufficient to hold all the Blue nodes at once. Let there be '*n*' such Blue configurations. The set of feasible blue configuration mappings is denoted as:

$$BC = \{BC_1, BC_2, ..., BC_n\} \tag{5.7}$$

This process has been implemented using MATLAB.

5.2 Player 2 - Red

Player 2: Player 2 is the attacker, denoted by Red. The attacker executes a DDoS attack on physical graph E. To disrupt the computer network, it places zombies on *E* nodes that attack nodes or arcs in E to disrupt Blue's distributed application by preventing Blue programs from communicating with each other. The attacker has limited resources, so it has to employ the minimum number of zombies and select the minimum number of nodes or arcs it has to disable to disrupt Blue connectivity.

Aim of Player Red:

1. Disrupt Blue connectivity by disabling maximum number of Blue configurations and forcing Blue into a position where it cannot reconfigure itself by entering a "loopy" [7] game.

Strategy: Red can disrupt a Blue configuration by placing zombies so as to either:

1. Attack Node Capacities
2. Flood Arcs

5.2.1 Attack Node Capacities

Red can place zombies on the same physical node on which there are one or more Blue nodes. No strategies are required for this condition as Red would know how much nodal capacity it has to consume in order to disable that Blue node. Hence it is crystal clear that Red would win this game if it had enough resources to consume the Blue node capacity. If Red cannot place zombies on the same physical node as the Blue then the situation becomes more challenging. How would Red know the minimum set of nodes it has to disable to disrupt the entire network for all possible blue configurations? This section explains how an attacker can determine the set of minimum number of nodes.

There are '*n*' Blue configurations out of which one of the Blue configurations is considered, say BC_1. There are '*M*' numbers of blue nodes. Each of these blue nodes can be a source and sink. So there will have '*M*' sources and '*M'* sinks. For all the possible combinations of blue source and sink nodes the min-cut is calculated. This min-cut will give a set of the 'special' edges removing which the source will be unable to communicate to the sink. The attacker will wish to further minimize this set of arcs to avoid wasting resources attacking the arcs that do not need to be attacked. This can be further explained in detail by an example.

In Figure 8, Blue needs a flow of 4 from *A* to B. Red has to disable both arcs with capacity 6 and 4 to disable blue connectivity. If the attacker disables arc 6 Blue can still send a flow of 4 over the other arc with the capacity 4. So in this case the Red has to attack both arcs 6 and 4.

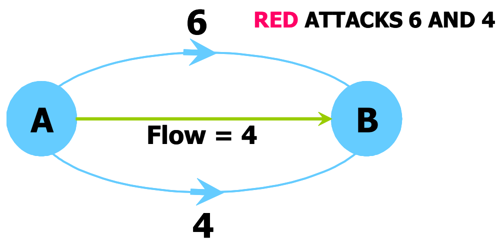


Figure 8. Reducing the number of arcs Red has to disable.

Now consider another scenario in which *A* has to send a flow of 5 to B instead of 4. If Red disables the arc with the capacity 4 Blue can stay connected but if Red disables the arc with the capacity 6, Blue will not be able to send a flow of 5 over the arc with a capacity of 4. So in this case disabling a single arc 6 will suffice. Red does not have to unnecessarily waste resources in disabling arc 4.

To accomplish this we carry out a few steps which will further reduce these arcs.

1. The set of edges in the min-cut are arranged in the descending order of their capacities.

2. Assume C_1, C_2, C_3……C_n are the capacities of these arcs and

$$C_1 \geq C_2 \geq C_3 \geq \text{............} \geq C_n \tag{5.8}$$

3. The sum of all the capacities in the min-cut which is denoted by A can be represented as:

$$A = \sum_{i=1}^{n} C_i \tag{5.9}$$

4. The slack (S) is then calculated by subtracting the Flow from A and is represented by:

$$S = A - Flow \tag{5.10}$$

5. Then a simple flow chart is used to find the set of minimum number of edges we need to disable for a particular min-cut. Figure 9 represents the flow chart for carrying out the above mentioned procedure.

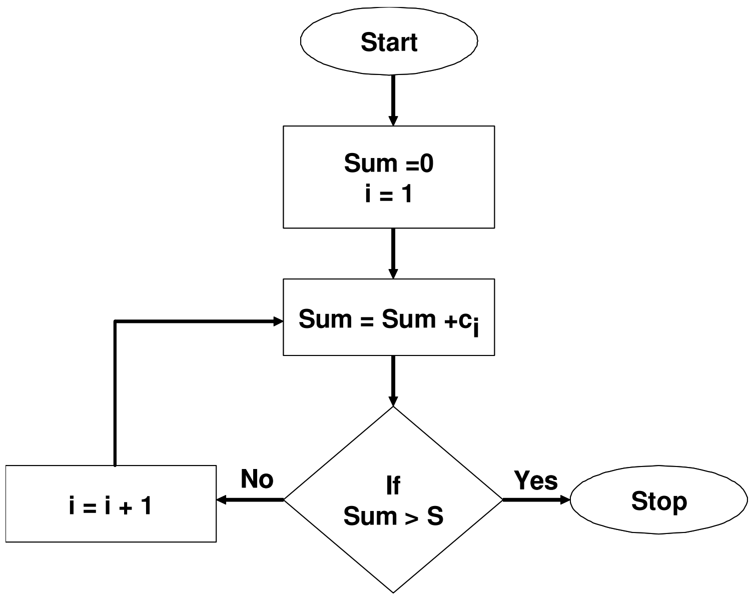


Figure 9. Flow chart for reducing the number of arcs in the min-cut.

To elaborate on this issue we find out the minimum number of arcs the attacker has to disable in Figure 8,

1. *Case 1:* When the flow is 4, C_1 = 6 and C_2 = 4. Here, A = 6 + 4 = 10 and the Slack(S) = A - Flow = 10 - 4 = 6. Sum = 0 + 6 = 6 and Sum is less than S so 'i' is incremented to 2. For the next iteration, the Sum becomes 6 + 4 = 10 which is greater than S, so we stop. Hence we deduce that we have to disable both arcs 6 and 4.

2. *Case 2:* When the flow is 5, C_1 = 6 and C_2 = 4. Here, A = 6 + 4 = 10 and the Slack(S) = A - Flow = 10 - 5 = 5. Also, Sum = 0 + 6 = 6 and Sum is greater than S, so we stop and deduce that disabling the arc with the capacity 6 is enough to disrupt Blue connectivity.

This gives a reduced set of arcs (R) but we require a set of nodes that we need to disable for a Node-attack. To implement this we take this set of arcs and for each arc we split the nodes of that arc into two nodes connected by unit capacity. For example consider an arc (1-2) whose nodes 1 and 2 are split as 1 and 1* and 2 and 2* respectively.

This is done for all the arcs in R. The entire graph is then rearranged with these split nodes incorporated into it. The split nodes are arranged in such a way that 1 has only incoming arcs and 1* has only outgoing arcs (1 and 1* actually mean the same node 1). Once this is done the min-cut for this graph is calculated which gives a set of arcs. All the split nodes are then merged back to get a min-cut of nodes. In a similar fashion, a set of nodes is found for each min-cut of BC_1. For the first min-cut of BC_1 the set of min-cut of nodes is A_1, for the second min-cut of BC_1 the set of min-cut of nodes is A_2 and so on till A_n.

After this, a set of minimum number of nodes common to all the Blue configurations that are absolutely necessary to disable the entire network is selected. For BC_1 we will have $A_1, A_2, A_3, \ldots\ldots, A_n$ sets. So for BC_1 if either one of the sets i.e. A_1 or A_2 or A_3 or…A_n is disrupted or disabled, Blue will not be able to communicate from a particular source to a sink in one of its configuration thus violating the Quality of Service [8], [38] and [53] for the respective Blue configuration.

A similar procedure is followed for BC_2, BC_3, …, BC_n. Consider three Blue configurations BC_1, BC_2 and BC_3 with set A_1 selected from BC_1, A_3 from BC_2 and A_7 from BC_3. In this case $A_1 \cap A_3 \cap A_7$ are needed to disable Blue. So the last step is to perform an AND operation on these sets of nodes corresponding to the Blue configurations. This will give us the minimum number of nodes we that we need to disable in order to disrupt Blue connectivity for any Blue configuration in BC. This is done by using a search tree with a Branch and Bound algorithm.

Branch and Bound search tree: A branch and bound algorithm is used to solve problems that have a finite but usually very large number of feasible solutions [54] and [55].

Statement: The problem is to minimize (maximize) a function *f*(*x*) of variables (x_1, x_2…..x_n) over a region of feasible values. The function f is called an objective function and may be of any type [54]. The set of feasible solutions is determined by general conditions on the variables. To solve the problem, two components need to be defined. The first one is branching by which the feasible region is covered by splitting into several smaller sub-regions. This procedure may be repeated recursively at each of the sub regions resulting in a search tree. The second component is bounding which is a fast way of finding the upper and lower bounds for the optimal solution. The search terminates when there are no unexplored parts.

An example showing the implementation of Branch and Bound search tree is demonstrated below. Consider four Blue configurations: BC_1, BC_2, BC_3 and BC_4. A_1, A_2 and A_3 are sets of nodes that we have to disable. The assumed values of for these sets of nodes are shown in Table 5.1. The tree is now built, starting with the minimum number of sets of nodes i.e., BC_4 as it has only one set of nodes i.e. A_1. The tree diagram is shown in Figure 10. The root of the tree has only a single node (Minimum number of nodes when traversing back the tree, M=1). The next Blue configuration we choose is BC_3 as it has only two sets: A_1 and A_2. The tree has two children where the first child has two nodes [2, 4] and the second child has a single node [1]. But since 4 is already there in the root. So the value of *M* for child 1 is still 2. The branch and bound search tree chooses the child with minimum number of nodes where the minimum number of nodes is *M*. Now in our case *M* is equal to 2 for both the options so one of the two children is randomly chosen. Suppose A_2 is chosen i.e. child 2.

Table 5.1 Blue configurations and min-cut nodes.

Set of nodes	BC_1	BC_2	BC_3	BC_4
A_1	1,2	1	2,4	4
A_2	2	2	1	
A_3	3	3,4,5		

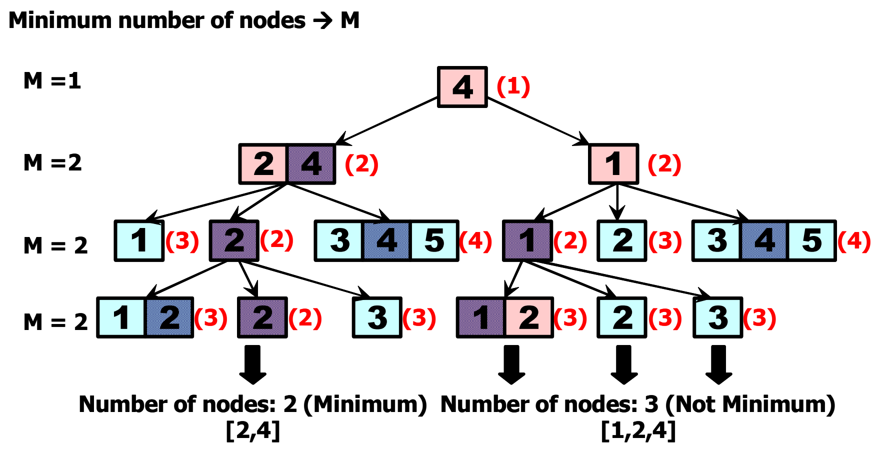


Figure 10. Branch and Bound Search Tree for Node Attack.

Now either BC_1 or BC_2 is chosen, as both have equal number of '*A*' sets. In our case, BC_2 is selected. In BC_2 the first child i.e. A_1 gives the minimum number of nodes as it does not add anything in the list of *M*. On traversing back it is seen that 1 has already been added to the list in level 2. Further considering the last Blue configuration BC_1, *M* becomes equal to 3. Now we have to cross check if any other option will give us a lower

M. As seen in Figure 10, if A_1 is selected instead of A_2, *M* is lower *M*. So this result is discarded and the one with the lower *M* is chosen. The above Branch and Bound search tree has been implemented in C language. On examining the input it is clearly seen that disabling set A_2 from BC_1, set A_2 from BC_2, set A_1 from BC_3 and set A_1 from BC_4 we can disable all the above Blue configurations.

For the node capacity attack it is typically difficult for Red to compromise the servers (nodes) used by Blue. When this does occur, Blue can also easily detect Red's presence and disinfect the server. We have discussed the node capacity attacks, their strategies and presented the results obtained using the tool in MATLAB but concentrated the focus of our analysis and simulations on flooding attacks as they are interesting, very likely to occur and difficult to prevent.

5.2.2 Attack Arc Capacities

In this attack, packets are used to flood links and exhaust arc capacities. The same question arises, like in the case of attacking the node capacities - How would the attacker know the minimum set of arcs it has to disable to disrupt the entire network for all possible Blue configurations? The approach used here is the same as used for attacking the nodes except that in this case we don't have to find arcs from nodes as we already get a set of arcs from the min-cut. So we follow the same procedure as used in the discussion of attacking node capacities. Table 5.2 shows the Blue configurations and the respective tree diagram using the branch and bound search tree is shown in Figure 11.

Table 5.2 Blue configurations and min-cut nodes.

Set of nodes	BC_1	BC_2	BC_3	BC_4
A_1	1-3,2-4	1-3	2-4,1-4	1-4
A_2	2-4	2-4	1-3	
A_3	3-4	3-4,1-4,4-5		

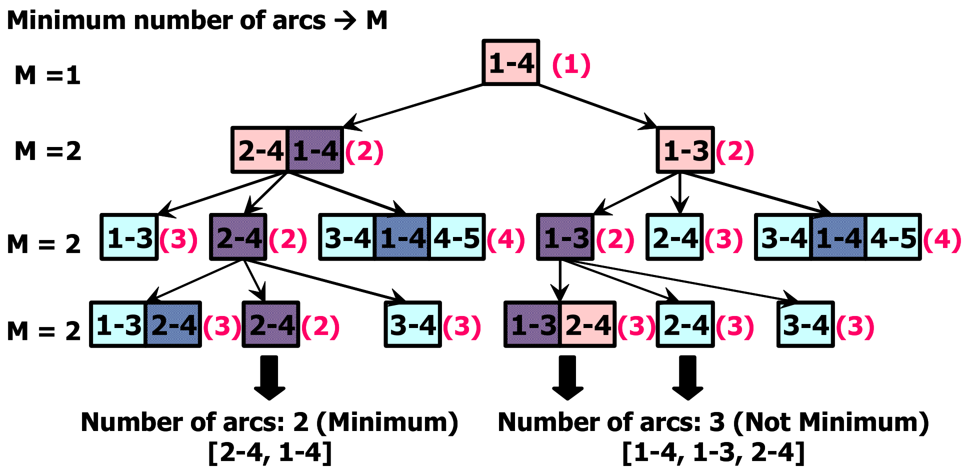


Figure 11. Branch and Bound Search Tree for Arc Attack.

On examining the input it is clearly seen that disabling set A_2 from BC_1, set A_2 from BC_2, set A_1 from BC_3 and set A_1 from BC_4 can disable all the above Blue configurations. The above Branch and Bound search tree has been implemented in C language.

5.2.3 Flooding the Arcs and Zombie Traffic

At this stage, the arcs that need to be flooded to disable Blue are known. The next step is to find out the amount of flow to be directed towards these arcs to disable them. In the example explained in Section 5.2.2, arcs (2-4) and (1-4) need to be disabled. Let the minimum flow to be directed towards these arcs to disable them be called as Red traffic which is denoted by ‘RT’.

RT → Red Traffic generated by the zombies

λ packets → Blue (Legitimate) traffic

C → Capacity of the physical arc to be attacked

Total traffic T is given by,

$$T = \lambda + RT \tag{5.11}$$

Traffic dropped D is given by,

$$D = (\lambda + RT) - C \tag{5.12}$$

Percentage of Blue (legitimate) traffic in the total traffic P is given by,

$$P = \lambda \div (\lambda + RT) \tag{5.13}$$

Expected rate of Blue (legitimate) traffic loss LTL is given by,

$$LTL = \lambda \div (\lambda + RT)[(\lambda + RT) - C] \tag{5.14}$$

The attacker will win i.e. will be successful in flooding the respective arc if,

$$LTL \geq \mathit{Blue\ Slack\ Traffic\ (BS)} \tag{5.15}$$

where Blue Slack traffic (BS) is given by,

$$BS = \mathit{Capacity\ \text{-}\ [Blue\ Flow]} \tag{5.16}$$

If a Blue arc does not have the available capacity to send the required flow it will try to send this flow on the other Blue arcs. So we have to check if the other blue nodes have a slack that can be used to send this flow. Hence, the Blue flow can be given by,

$$\mathit{Blue\ Flow} = \mathit{[Blue\ Capacity\ \text{-}\ Slack\ of\ other\ Blue\ Nodes]} \tag{5.17}$$

Therefore,

$$BS = \mathit{Capacity\ \text{-}\ [Blue\ Capacity\ \text{-}\ Slack\ of\ other\ Blue\ Nodes]} \tag{5.18}$$

Using equation (5.14) and (5.15) we get,

$$BS \geq \lambda \div (\lambda + RT)\,[(\lambda + RT) - C\,] \tag{5.19}$$

Blue Slack traffic (*BS*) should be at least equal to LTL. Solving for *RT*,

$$\frac{BS \times (\lambda + RT)}{\lambda} = [(\lambda + RT) - C]$$

$$RT = \frac{C}{[1 + (BS \div \lambda)]} - \lambda \qquad (5.20)$$

(5.20) gives us the red traffic an attacker needs to generate in order to disable a Blue arc. Also in the above equation, Blue Slack is assumed to be equal to LTL (5.19) which need not be true. So 'traffic' little more than *RT* i.e. → (*RT* + 1) is needed to disable the arc. Also in the above case *BS* is assumed to be less than λ. If a remainder of zero is obtained i.e. when *BS* = λ, then an infinite amount of flow is needed to disable that arc.

5.2.4 Zombie Placement

The vulnerable nodes and arcs and the amount of flow to be routed to the arcs are now known. The final step is to find the source of Red traffic to be generated i.e. Zombie Placement.

Steps for finding optimal zombie positions,

1. In every Blue configuration for each physical source and sink we have the max-flow and the min-cut. Calculate *RT* for each min-cut using (5.20). The *RT* is calculated for a single arc and not the entire min-cut. It is very easy to convert the *RT* for a single arc to *RT* for a min-cut. For the capacity *C* use the sum of the physical capacities over all the arcs in the min-cut, λ is Blue traffic over that min-cut. Blue Slack can be a little difficult to calculate. For simplicity, we will ignore Blue interfering with its own traffic. So equation (5.18) becomes,

$$BS = \mathit{Capacity} - [\mathit{Blue\ Capacity}] \qquad (5.21)$$

where Slack of other blue nodes is assumed to be zero.

2. Consider the physical nodes without the Blue nodes as sources (as we do not want to place the zombies on the same node as the Blue node). Sinks can be all the physical nodes. A good source candidate will be close to the Blue source and a good sink candidate will be close to the Blue sink.

3. Check if the max-flow for any of the sources we selected is greater than RT of any one min-cut in every Blue configuration. For a Blue configuration if any one value of RT < max-flow, select that node.

4. Repeat the above steps for all the Blue configurations.

5. Pick up a common zombie node in all the Blue configurations. If we do not find a single zombie node we have to look for two or maybe more zombie nodes. We use the Branch and Bound search tree [54], [55] to calculate the minimum number of zombie nodes common to all the Blue configurations. The above Branch and Bound search tree has been implemented in C language.

In this way the optimal zombie positions to execute an Arc Attack are determined.

CHAPTER 6

DISTRIBUTED DENIAL OF SERVICE MITIGATION APPROACH – RECONFIGURATION STRATEGIES

6.1 Game

We now know the number of zombies required to disrupt all the Blue configurations in BC but what if the attacker does not have enough zombies to disable all the Blue configurations but lesser number of zombies that will only disable some of the Blue configurations in BC? This gives Blue a chance to reconfigure from the DDoS attack from a particular Blue configuration that is attacked to another Blue configuration that Red cannot attack. This way we can have a board game set up. We know explain a few terms like the surreal numbers and Combinatorial Game Theory that are required for further understanding of the material discussed in this section.

6.1.1 Surreal Numbers and Combinatorial Game Theory

Surreal Numbers: A surreal number [6], [56], [57], [58] is a pair of sets (Left set and Right set) of previously created surreal numbers. No member of the Right set may be less than or equal to any member of the Left set. Also a surreal number x is less than or equal to a surreal number y if and only if y is less than or equal to no member of x's left set, and no member of y's right set is less than or equal to x.

Please refer to [57], [58] and [59] for further information on surreal numbers.

To understand surreal numbers in detail some examples of how real numbers can be represented as surreal numbers are given below.

$$0 \equiv \{ \mid \} \rightarrow \text{'}\equiv\text{' is to represent equality in surreal numbers}$$

$$1 \equiv \{ 0 \mid \} \rightarrow \text{as 1 is greater than 0}$$

$$-1 \equiv \{ \mid 0 \} \rightarrow \text{as -1 is less than 0}$$

Similarly we can represent 2 and -1 with the above equations,

$$2 \equiv \{ 1 \mid \} \text{ and } -2 \equiv \{ \mid -1 \}$$

Also we can define numbers like ½ and -½ in the form of surreal numbers,

$$\tfrac{1}{2} \equiv \{ 0 \mid 1 \} \text{ and } -\tfrac{1}{2} \equiv \{ 1 \mid 0 \}$$

Also a real number does not necessarily be represented in surreal numbers in a particular way; there are multiple representation ways in surreal numbers for real numbers. For example,

$$2 \equiv \{ -1, 0, 1 \mid \}$$

$$2 \equiv \{ 0, 1 \mid \}$$

$$2 \equiv \{ -1, 0, 1 \mid \}$$

$$\text{and } 2 \equiv \{ 1 \mid \}$$

We now know these surreal numbers: −2, −1, −½, 0, ½, 1, and 2. Again, we can create new surreal numbers based on these.

Combinatorial Game Theory: The definition of surreal numbers contains one restriction that each element of the Left set must be strictly less than each element of the Right set. If this restriction is dropped we can generate a more general class known as *games*. A combinatorial game [60] typically involves two players, called Left and Right and the corresponding pair of sets – Left set and Right set. Addition, negation, multiplication, and comparison are all defined the same way for both surreal numbers and games. Every surreal number is a game, but not all games are surreal numbers. There are several types of these two-player perfect information games. A particular one that is of interest with respect to this thesis is the one that does not result in any ties and has one of the four outcomes, player Left wins, player Right wins, the first player to move wins or the second player to move wins [60]. A mathematical theory has been developed for analyzing the strategies a player will use in order to win the game using game trees. A game tree has a root node that is the starting point of the game. The root node has zero or left branches (options) for the Left player and zero or right branches for the Right player i.e. moves for Right player are represented by edges that go down and right and moves for Left player are represented by edges that go down and left. Each player sees the options it has and makes the move that maximizes his gain. Game trees can be represented systematically as a generalization of surreal numbers in the form:

$$\{L_1 \ldots L_n \mid R_1 \ldots R_m\} \tag{6.1}$$

where Left can choose any move from L_1 to L_n and Right can choose any move from R_1 to R_m. Every element L_i and R_i of Equation (6.1) is either a numeric value or a recursive surreal number in the form of Equation (6.1). If;

$$\forall L_i \forall R_j : L_i < R_j \tag{6.2}$$

then Equation (6.1) has a unique numeric value. The value of a surreal number where Condition (6.2) holds is the "simplest" number between the greatest *L* value (L_{max}) and smallest *R* value (R_{min}) [59]. Typically, the simplest number has the value $i + \frac{j}{2^k}$ in range ($L_{max} \ldots R_{min}$) where *i, j,* and *k* are integers and *k* is a minimum. If Condition (6.2) is not the case, i.e.:

$$\exists L_i \exists R_j : L_i \geq R_j \tag{6.3}$$

Then the number is ill formed. It represents a game and the value of the game depends on the sequence of moves taken. Consider a simple example of a game tree shown in Figure 12. This game tree can be represented in surreal numbers as;

$$G = \{15, \{10 | 5\} || -8\}$$

If Right plays first then he has only one option, so he gains 8 points from the Left player. If Left plays first, he has two options, either to collect 15 points from Right or move to the game {10 | 5}. If he moves to the game {10 | 5}, Right plays next and gives player Left 5 points. Player Left would prefer to gain 15 points rather than 5, so if Left plays first he chooses to collect 15 points.

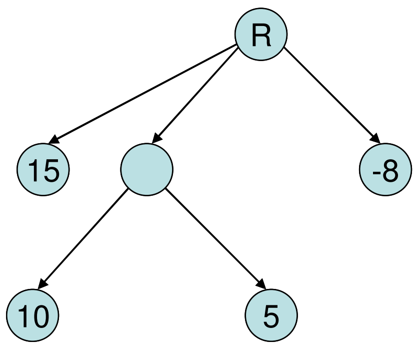


Figure 12. Game tree representation for G.

These games have the following rules [6]:

1. There are just two players, called Left and Right
2. There are several positions and a starting point.
3. There are clearly defined rules that specify the moves that either player can make from a given position to its options.
4. Left and Right move alternately, in the game as a whole.
5. The player unable to move looses.
6. Both players know what is going on, i.e. complete information
7. The rules are such that the play will always come to an end, as some player will be unable to move.

For this thesis we have modified the rules of the game to best suit the options available and they are given in Section 6.1.2. Before we jump to the rules of the game there is one key concept that needs to discussed and that is a *loopy game*.

Loopy Games: Combinatorial game theory usually assumes no position may be repeated. The games we discuss may involve repetition. A *loopy game* [7], [61] and [62] is one that allows repeated positions. Every game that permits repetition faces the possibility of non-terminating play i.e. cycles in the graph. For these games the stopping conditions are required and need to be pre-defined for the game in question. This is typically resolved by declaring infinite plays as a "draw game" as in Chess. But there can be other stopping conditions, like Hare and Hounds where for infinite plays Hare is the winner [6], [61]. For the DDoS games, we allow repetition (loopy games). The rules for the game and the pre-defined stopping conditions are discussed in the next sub-section. For more information on loopy games refer to [61] and [62].

6.1.2 Example Game

Rules of the Game: Based on the Combinatorial Game Theory we now define the rules for the two players of the DDoS game – Red i.e. Right Player and Blue i.e. Left Player before they actually begin to play the game.

Rules for Blue:

1. Blue always starts the game.
2. Blue is allowed only one move at a time.
3. Blue can select one possible configuration out of the available Blue configurations. Blue chooses a configuration that is not currently disabled by Red.
4. Blue cannot have redundancy i.e. multiple Blue copies.
5. Blue reconfigures by moving a single process from a physical node to another i.e. Blue can move the position of only one nodes.

6. Blue cannot move to a configuration that Red can attack using the present set of zombies.

7. Blue will try to find a loopy game [7], [61] and force Red into it where it can always move to the previous configuration. If Blue succeeds in creating a "loopy" game where Red cannot escape from the loop, it wins since it can recover from any attack.

8. Blue has perfect knowledge of Red's zombie positions.

9. Stopping Condition: If Blue cannot make a move to any one of its options that is not under attack by Red, Blue loses.

Rules for Red:

1. Red is also allowed one move at a time.

2. Once Red places a zombie on a particular node it cannot move that zombie until its next turn.

3. Red tries to force Blue into a position where it cannot reconfigure itself.

4. Red tries to choose a zombie or a set of zombies that can affect maximum elements of *BC* (as he does not have enough zombies to affect all the elements of *BC*).

5. Red has perfect knowledge of the Blue configurations and what configuration Blue has chosen for the current move.

6. Stopping Condition: If Red is unable to find a zombie to attack the current configuration that Blue is in or if Blue forces Red into a loopy game Red looses.

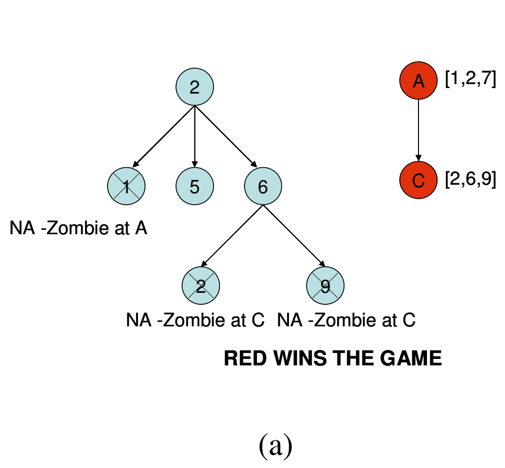
Let's consider an example. The moves of the Blue player are the configurations it can reach from the current configuration. Our example has 3 Blue nodes and 6 physical nodes. The MATLAB tool has given us 10 Blue configurations say [1, 2, 3………10] that satisfy all the constraints and we require 5 zombies to disable all ten Blue configurations. But due to limited resources Red can only use 2 zombies at a time. Each combination of 2 zombies is a Red move denoted by *A*, *B*, *C*, …., and so on. Table 6.1

shows the details of the game. We assume that Blue starts the game with Blue configuration 2 and in response to that move Red can either choose *A* or *C* (as both the zombie moves – *A* and *C* can disable Blue configuration 2).

Table 6.1 Details of an example Game

Blue configuration	Reconfigure	Zombie Move	Zombies	Disrupt Blue configurations
1	2, 3, 4	A	3, 5	1, 2, 7
2	1, 5, 6	B	1, 6	3, 4, 5
3	1, 7	C	3, 4	2, 6, 9
4	1,10	D	1,5	8, 9, 10
5	2, 8, 9			
6	2, 9			
7	3, 10			
8	5			
9	5, 6			
10	4, 7			

Let's assume that Red chooses *A*. The game tree for the above example would look like Figure 13. If we have to denote these game trees in the form of surreal numbers, the first level would look like {1, 5, 6 | *A*, *B*, *C*, *D*} where 1, 5, 6 are moves Blue can make (from Blue configuration 2) and *A*, *B*, *C*, *D* are moves Red can make. As we can see in Figure 13.a at level 2 if Blue chooses configuration 6 it will loose but if it chooses configuration 5 it has a stronger chance of winning. If Blue can form a loop as shown in Figure 13.b [2-5-2] and [5-8-5] Blue will never loose as it can keep looping between these two nodes and Red will never be able to disrupt it as it will keep reconfiguring itself. In 1a we can see that Red has led Blue into a position where Blue is unable to reconfigure itself so Red wins the game.



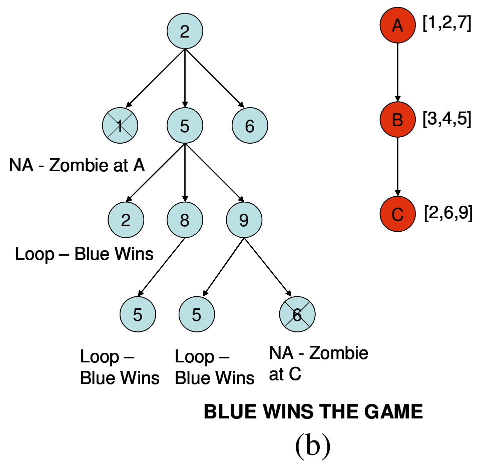


Figure 13. Game trees (a) Red Wins (b) Loopy Game – Blue Wins
(NA – Not Allowed)

6.2 Sum of Games and Thermographs

In practice, any given enterprise relies on multiple distributed processes. So, rather than Blue having one distributed application, it can have multiple distributed applications. Similarly, an attacker can not expect to destroy all of the processes used by the enterprise at any point in time. The attacker will try to maximize the number of processes it can disable at any point in time. This situation describes a "sum of games" problem [6], where Blue and Red alternate moves. We have modified the rules for the "sum of games" problem to best suit the options available and they are given in Section 6.2.1. The payoff's for the end nodes in the Blue configurations is the smallest slack bandwidth on a min-cut of the configuration i.e. the remaining slack on the physical arc after Red has attacked.

Sum of Games: Each player plays a set of games (distributed process), G_i. At each turn, the player chooses a game i.e. a distributed process out of the multiple distributed processes and a move to make in that game in order to maximize the value of the game. This sum is a game G_i, such that,

$$G_i = \sum_{j=1}^{n} G_{i,j} \tag{6.4}$$

We consider the value of the sum of games to determine the best move for the player according to his needs. Of particular importance are the following results (proofs in [56]):

Theorem 1: Calculating the value of a sum of games is NP-hard.

Theorem 2: Finding the optimal sequence of moves for a sum of games is PSPACE-complete.

Convincing example game trees for these assertions are in [56]. These theorems state that a truly optimal strategy for a sum of games is only found by an exhaustive search of the alternatives. This requires exponential time and is unsuited for non-trivial problems. Instead of finding the best possible solution it is possible to find approximate solutions within a constant offset of optimal [56]. Though this problem has been shown to be P-Space complete [56] Berlekamp has used thermographs, to tractably find near optimal solutions.

In this section we introduce a concept called thermographs that show how chilling the surreal number game representation in the form of Equation (6.1) can find approximately optimal strategies for sums of games. In a really complicated battle or a hot game [6], a player will have a difficult time in deciding what move to make so he has to use some sort of strategy to cool the game where he can easily make a decision.

Thermographs: Thermographs are use to calculate the value of a game. The variability of a game is its temperature. When all L and R values in Equation (6.1) are atomic and Condition (6.3) is true, the temperature can be computed by averaging the negative of the smallest R value with the largest L value. This is the amount that stands to be gained by either player initiating a move. A game where much (little) stands to be gained or lost is *hot* (*cold*) [6]. We use the relative temperatures of the component games $G_{i,j}$ to decide which game in Equation (6.4) to play in at any point in time.

The game temperature comes from the insight that the mean value of a game may be kept constant, but the variability reduced if a tax t were imposed for making a move. This process (cooling) is done by modifying the game G_t:

$$G_t = \{G_t^L - t \mid G_t^R + t\} \tag{6.5}$$

The coordinate system used in drawing thermographs has the tax on the y-axis and game value on the x-axis [63]. The values on the x-axis are in descending order to keep Left's moves to the left and Right's move to the right.

As tax t increases, both sides reach a common value that is the game's *mean value* [56]. The smallest tax needed to reach the game's mean value is its *temperature* or *freezing point*. For each game that does not contain a possible loop, its thermograph ends in an infinite vertical mast. Generalized thermographs for games that contain loops are described in [7].

Figure 14 is a thermograph for game {2 | -1}. This game reduces to a mean value of ½ when it is taxed (cooled) by any value over 1½. The temperature t=1½ is the *freezing point* of this game.

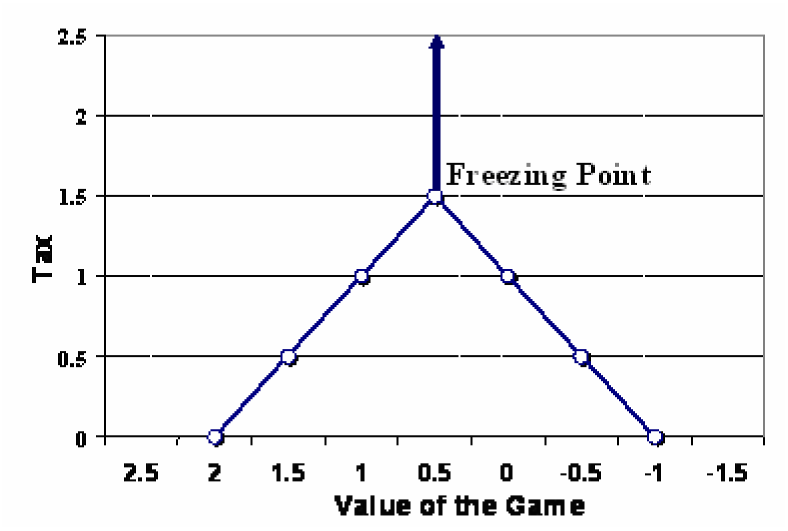


Figure 14. Thermograph of the game {2 | -1}.

To plot a thermograph, start with the atomic games where Left and Right's choices are numbers and recurse upwards. For example, Figure 15 shows the thermograph of {{5 | -5} | -20}. First plot the thermograph of {5 | -5} by marking the Left and Right choices for t=0 on the horizontal axis then plotting the game values as t increases until the Left and Right values converge [64]. Since the value on the right is already a number (-20), its thermograph is just a vertical mast.

The next step is to plot the thermograph of {{5 | -5} | -20} using the thermograph of {5

| -5}. After Left has moved to {5 | -5} it will be Right's turn so -5 is the starting point on the left. The temperature of the freezing point of {5|-5} is 5. So the left edge of the thermograph starts at point (-5, 5).

The game -20 has value -20 and freezing point *t*=0. So the right edge of the thermograph starts at point (-20, 0). We follow Equation (6.5) by subtracting a tax *t* from the left and adding it to the right, until the two values converge. As shown in Figure 15, this gives us the freezing point (temperature) of 10 and a mean value of -10. More examples can be found in [6].

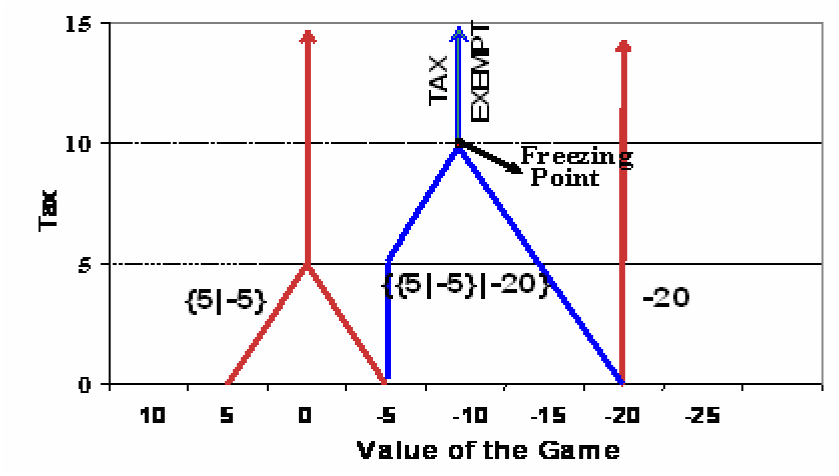


Figure 15. Thermograph for {{5 | -5} || -20}.

The thermograph allows us to simplify the game by summarizing a complex game tree into two aspects: (*i*) the temperature summarizes the importance of a game by stating the amount of variability, and (*ii*) the range of values at a temperature state (distance from the Left to the Right mast of the thermograph) shows how much stands to be gained at that temperature. Note that where a mast exists, no side stands to gain anything by playing that game.

Thermograph based strategies: The use of thermographs for determining game strategies is described in detail in [7] and [6]. The main concept of this approach is "chilling" or reducing the temperature of the game represented in the thermograph [65].

By starting with a high temperature (tax) and considering only games where something stands to be gained at that temperature, it is possible to dramatically reduce the search space of the problem. Of particular interest is this result [56].

To find the sum of games and make an optimal move in that game we need to add thermographs, but we cannot just add thermographs [6], [64], [62]. When there are too many components, the optimal strategy can be very complex and time and computing resources required determining the best move would be substantially magnanimous.

*Theorem 3***:** For any sum of games, it is possible to find a strategy that attains the optimal value of the sum to within the value of the second most valuable game.

In [64], three strategies are proposed that use temperature *t* to find near optimal strategies within the bounds of theorem three. Assuming that the game has a current tax rate *t* active, choose from among the active games the one that:

- *Hotstrat* – has the maximum temperature,
- *Thermostrat* – has the largest difference in value between the left and right bound at tax *t*.
- *Sentestrat* – is the region where your opponent just moved.

In the majority of cases, these three strategies are equivalent. But there are cases where hotstrat behaves poorly.

To determine the optimal move for the DDoS games we plan to use the thermostrat strategy as thermostrat makes a million optimal moves and a few sub-optimal moves [6]. The same thing cannot be stated about the other two strategies.

Thermostrat: To understand thermostrat, we will make the reader go through an example. Consider a game *C* that is made up of *A+B*. There can be many such games but we illustrate two games for simplicity. We need to find an optimal move for player Left (Blue), in the games *A* and *B*. First we draw the individual thermographs for A and B. Figure 16 shows the individual thermographs for game *B* and game *A* where *B* = {12|6} and *A* = {{8|4}|{-4|-8}.

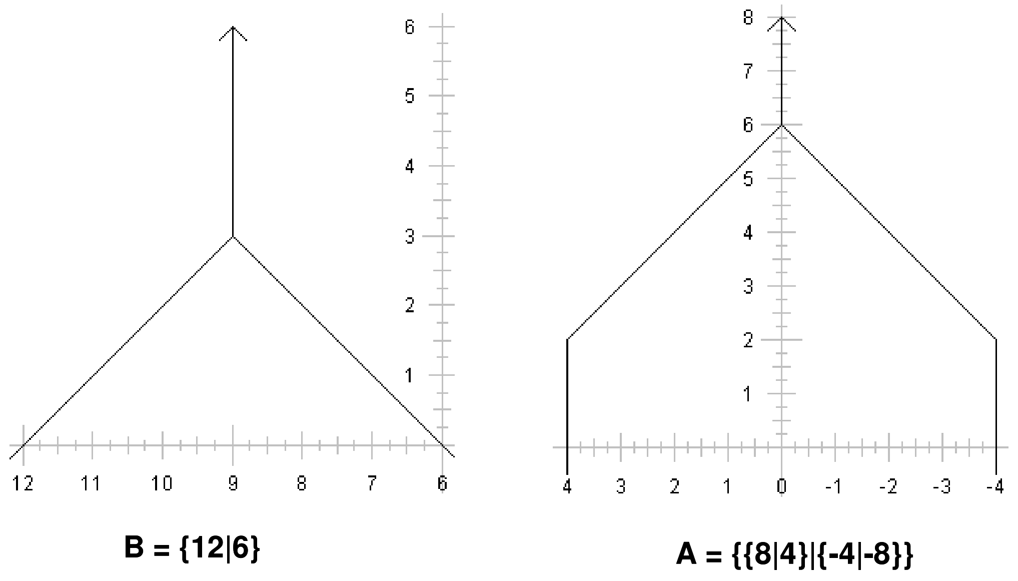


Figure 16. Thermographs for *B* and *A*.

The next step for the Left (Right) player is to add the Right (Left) boundaries of the individual thermograph at each tax level to get the compound thermograph. So the right boundary of the compound thermograph is given by;

$$R_t(C) = R_t(A) + R_t(B) \tag{6.6}$$

This behavior is demonstrated in Table 6.2.

Table 6.2 Right boundaries for compound thermograph at each tax interval for Left (Blue)

Tax Level (t)	$R_t(A) + R_t(B)$	$R_t(C)$
0	6 + (-4)	2
1	7 + (-4)	3
2	8 + (-4)	4
3	9 + (-3)	6
4	9 + (-2)	7
5	9 + (-1)	8
6	9 + (-0)	9
7	9 + (-0)	9

We then calculate the width of the compound Thermograph, by taking the maximum width of *A* and *B* at each tax level. So the maximum width (W_t) at each tax level is given by;

$$W_t = max\{W_t(A), W_t(B)\} \tag{6.7}$$

This behavior is demonstrated in Table 6.3.

Table 6.3 Maximum Width at each tax interval

Tax Level (t)	Width of A	Width of B	Maximum Width
0	8	6	8
1	8	4	8
2	8	2	8
3	6	0	6
4	4	0	4
5	2	0	2
6	0	0	0
7	0	0	0

The next step is to add the Maximum Width and the Right boundaries at each interval.

Table 6.4 Adding Maximum Width and the Right Boundary of each game

Tax Level (t)	Max Width + $R_t(C)$
0	8 + 2 = 10
1	8 + 3 = 11
2	8 + 4 = 12
3	6 + 6 = 12
4	4 + 7 = 11
5	2 + 8 = 10
6	0 + 9 = 9
7	0 + 9 = 9

The maximum value of Game C is 12 and it is 12 at tax level 2 and 3. So the temperatures at which Left feels most comfortable are $T = 2$ and $T = 3$. Since Left would try to be safe and prefer only as much as heat as is absolutely necessary, he chooses the minimum ambient temperature which is $T = 2$. In [6] this concept is very nicely summarized as;

The ambient temperature is the least T for which $R_T(A) + R_T(B) + W_T$ is maximal. The component that is widest at $T = 2$ is A so player Left should make a move in component A.

6.2.1 Example

We now present a working example consisting of 3 distributed Blue applications. For determining the process in which the move has to be made, we make use of Thermostrat strategy.

Rules of the Game (in addition to the previous rules):

1. The payoff's for the end nodes in the Blue configurations is the remaining slack on the physical arc after Red has attacked. This can be given as;

$$\textit{Remaining Slack} = \textit{Capacity of Physical Arc} - (RT + BS) \qquad (6.8)$$

2. If Red is successful in placing a zombie(s) for a particular configuration, then this left-over slack will most definitely be a negative value. Blue will try to maximize this Left over slack and have the best performance for users and Red will try to minimize it and cause the users to suffer the most. For the game, we would like to determine not only who would win or loose but who would win or loose and by how much.

3. So if the two players have two options of -1 and -4. Red will choose -4 and Blue will choose -1.

4. Blue reconfigures by moving a single process from a physical node to another also since we have a directed graph to move a process from one node to another, there should be a direct link connecting the nodes. So there might be nodes a player can move a process to, but may not be able to move the process back to the node. This gives the problem a very practical approach as Blue will not always have loopy games at every position.

5. Loopy Games [61], [64] are incorporated in the game and one can specify which player (Left or Right) will win the game if the game is loopy. In our case if Blue is able to find a loopy game, it will most certainly win that game.

6. The Combinatorial Game Suite (CGSuite) [61] is an open source program built to aid research in combinatorial game theory. Given the values, it can be used to plot thermographs. It also has added functionality and full support for loopy games. Also in CGSuite one can specify which player wins the game if the game is loopy. Entering a game in CGSuite is very simple, its just entering it in the surreal number form like $G = \{2|-4\}$. To plot the thermograph of G, one just has to type Plot(Thermograph(G)). Entering loopy games is a little difficult. If we have $A = \{B|C\}$ and $B = \{A|D\}$. Now if we have to enter the game A, one has to put in, $P{:=}A{:}\{B{:}\{|A||D\}|C\}$. Loopy games will not be simplified automatically unless they are stoppers. To simplify loopy games, we have to input Sidle(P). Also if we want the Left (Right) player to win when the game is loopy, we will type Onside(P) (Offside(P)). We can plot both the thermographs on one plot using KoPlot(P).

Now we present the assumed values for the 3 Blue distributed application:

1. 1st Distributed Application: The first distributed application has 6 Blue configurations and it has loopy game. Game 1 is represented in Figure 17.

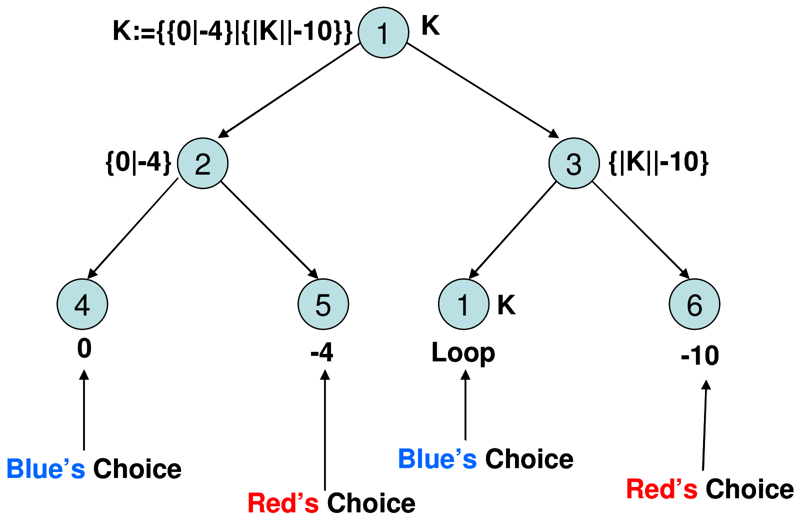


Figure 17. Game Tree for distributed application 1 – Game 1.

2. 2nd Distributed Application: The first distributed application has 10 Blue configurations and it has loopy game. Game 2 is represented in Figure 18.

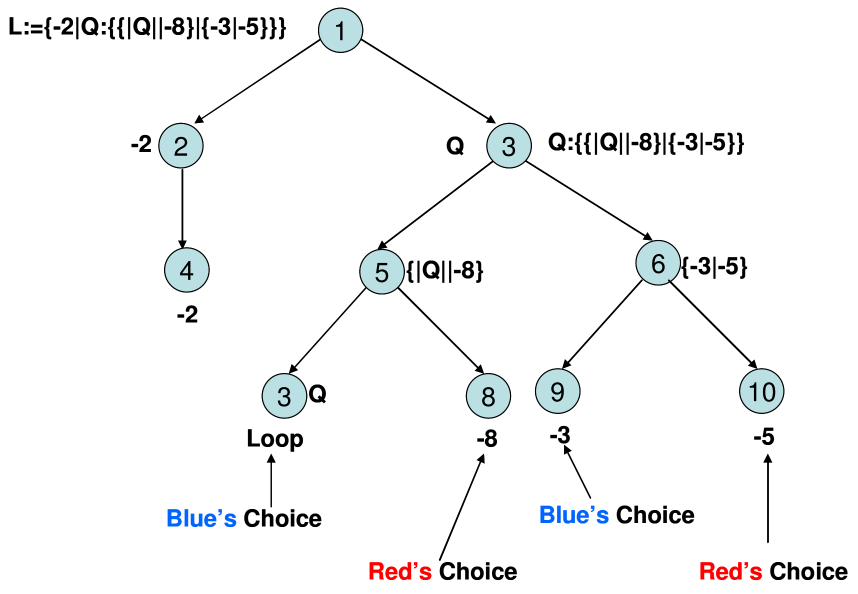


Figure 18. Game Tree for distributed application 2 – Game 2.

3. 3rd Distributed Application: The second distributed application has 12 Blue configurations and it does not have a loopy game. Game 3 is represented in Figure 19.

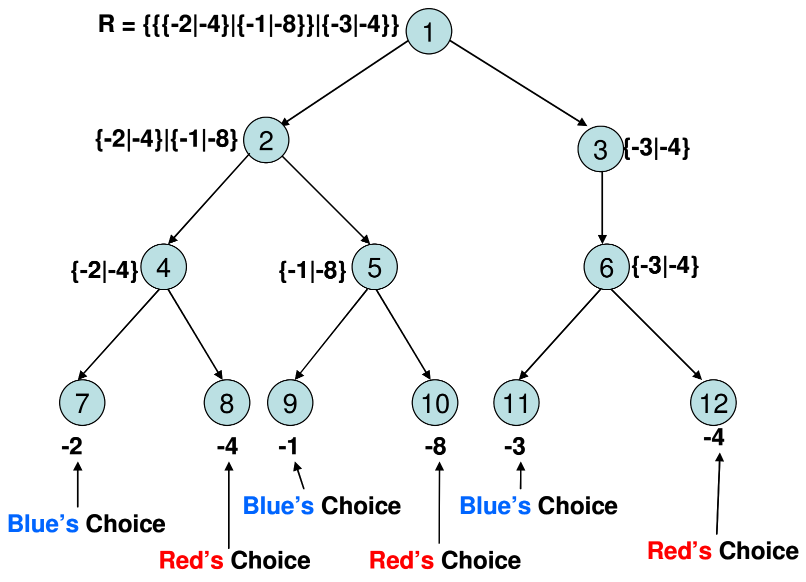


Figure 19. Game Tree for distributed application 3 – Game 3.

The next step is to use the Thermostrat strategy to decide which game Blue chooses and makes a move in it in order to cause least difficulty to the users. Like the example used to explain the thermostrat strategy, we will solve this example in a similar fashion. Figure 20 represents the thermographs of each of these games. Tables 6.5, 6.6 and 6.7 represent the calculation done on these thermographs to find the compound thermograph. Since Blue is the Left player we first have to find the sum of all the Right boundaries at respective tax levels (Table 6.5). Table 6.6 shows the maximum width (W_t) at each tax level and the finally in Table 6.7 we add the Maximum Width and the Right boundaries at each interval to find the ambient temperature.

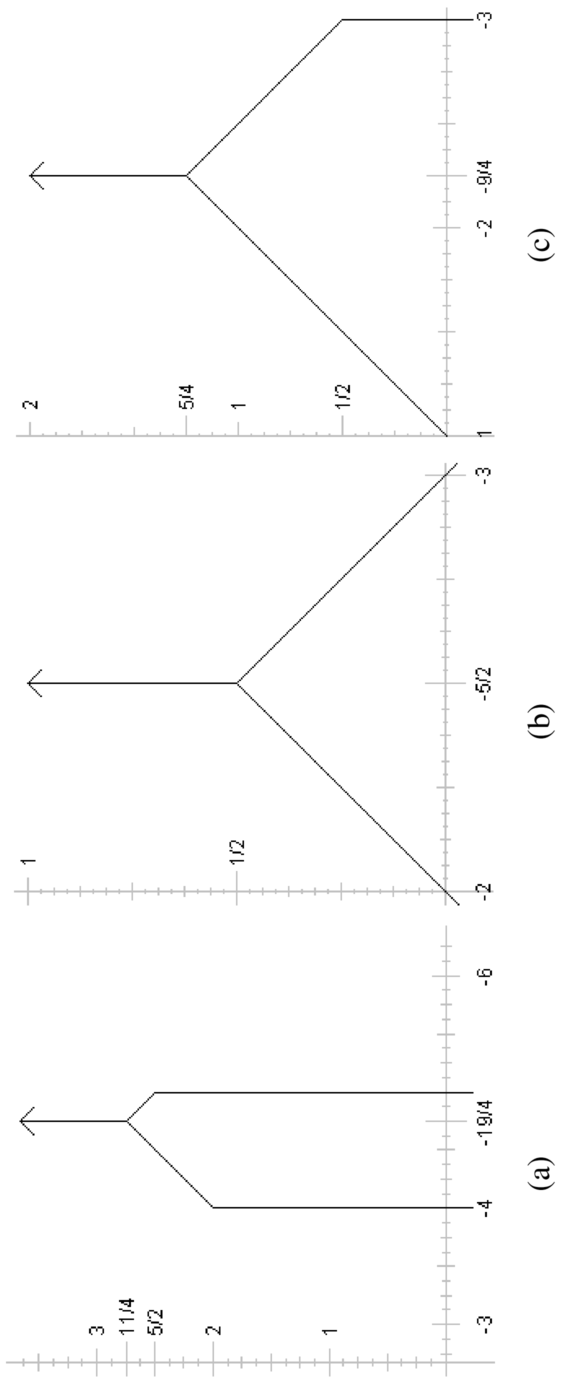


Figure 20. Thermographs to calculate sum of games (a) Game 1 (b) Game 2 (c) Game 3

Table 6.5 Right boundaries for compound thermograph at each tax interval for Left (Blue)

Tax Level (t)	$R_t(K) + R_t(L) + R_t(R)$	$R_t(G)$
0	(-5) + (-3) + (-3)	-11
0.5	(-5) + (-2.75) + (-3)	-10.75
1	(-5) + (-2.25) + (-2.5)	-9.75
1.5	(-5) + (-2.25) + (-2.25)	-9.50
2	(-5) + (-2.25) +(-2.25)	-9.50
2.5	(-5) + (-2.25) +(-2.25)	-9.50
3	(-4.75) + (-2.25) +(-2.25)	-9.25

Table 6.6 Maximum Width at each tax interval

Tax Level (t)	Width of K	Width of L	Width of R	Maximum Width
0	1	1	2	2
0.5	1	0	1.5	1.5
1	1	0	1	1
1.5	1	0	0	1
2.0	1	0	0	1
2.5	0.5	0	0	0.5
3.0	0	0	0	0

Table 6.7 Adding Maximum Width and the Right Boundary of each game

Tax Level (t)	Max Width + $R_t(G)$
0	-11 + 2 = -9
0.5	-10.75 + 1.5 = -9.25
1	-9.75 + 1 = -8.75
1.5	-9.5 + 1 = -8.5
2	-9.5 + 1 = -8.5
2.5	-9.5 + 0.5 = -9
3	-9.25 + 0 = -9.25

The maximum value of Game *G* is -8.5 and it is -8.5 at tax level 1.5 and 2. So the temperatures at which Left feels most comfortable are $T = 1.5$ and $T = 2$. Since Left would try to be safe and prefer only as much as heat as is absolutely necessary, he chooses the minimum ambient temperature which is $T = 1.5$. In [6] this concept is very nicely summarized as;

The component that is widest at $T = 1.5$ is A so player Left should make a move in component K. Also notice that K has a loopy game and Blue would definitely benefit by making a move in this game.

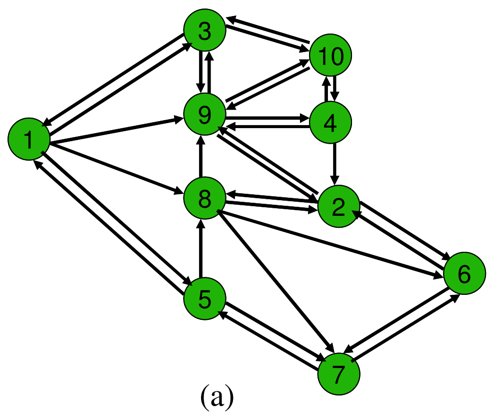
CHAPTER 7

SIMULATIONS AND RESULTS

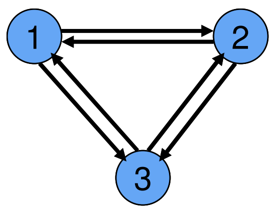
This chapter includes results from the MATLAB tool and the simulations done on a network using SSFNet simulator [66]. The MATLAB tool is used to find the possible Blue configurations for Blue and the zombie places for Red, for a given network and a distributed application. In order to verify if the tool correctly calculates the Blue configurations, the zombie places and if the calculated zombie places are minimum in number we perform a few simulations using SSFNet.

7.1 Results from MATLAB Tool

Figure 21 represents a physical network of 10 physical nodes and 30 arcs. The distributed application comprises of 3 Blue nodes. Figure 21.a is the physical network and Figure 21.b is the distributed application.



(a)



(b)

Figure 21. Input Networks (a) Physical Network (b) Blue Network.

The connectivity matrices *EE* and *BE*, where the capacities are in units of Mbps, are given by;

$$EE = \begin{bmatrix} 0030100510 \\ 0000010110 \\ 1000000012 \\ 0100000012 \\ 1000002100 \\ 0100002000 \\ 0000210000 \\ 0100011010 \\ 0221000001 \\ 0021000010 \end{bmatrix}, \quad BE = \begin{bmatrix} 012 \\ 201 \\ 210 \end{bmatrix} \tag{7.1}$$

The exact input and the output of the MATLAB tool are in appendix A for further reference. The results give 70 possible Blue configurations for the network shown in Figure 21.

For a DDoS attack an attacker can deploy thousands of zombies so we had to consider a very large network to give the attack a more realistic approach. So the number of physical nodes was increased to 500 and about 1500 arcs connected these nodes. The number of Blue nodes was increased to 5. These results are not attached to prevent repetition.

7.2 SSFNet Simulations

To verify that if the Blue configurations are generated correctly and the number of zombies and generated traffic *RT* are both minimum few verifications are performed on the generated results.

Verification 1: Blue nodes are randomly placed on physical nodes (other than current Blue hosts) with large capacities. This is to verify if the physical nodes that are discarded are short of bandwidth and possibly cannot host the Blue nodes. When SSFNet simulations were performed for these, some Blue (TCP) connections did not show any packet drops. But for every simulation at least one or more TCP connection exhibited packet drops. This would be unacceptable as this would violate the Quality of Service requirements for that Blue configuration.

Verification 2: Next we verify if the number of zombies obtained through the MATLAB tool is minimal. For this a random number of zombies and random places were chosen. Also zombie places close to the Blue sources were checked as places that would be good candidates for Red zombies. Randomly placed zombies did cause a DDoS attack, but the number of zombies required to do the attack were greater than or equal to the number of zombies generated by the MATLAB tool. The example we discuss in the next section does not have zombies close to the Blue sources but still manages to cause a successful DDoS attack.

For analysis, we assigned different colors to the hosts in SSFNet. The normal clients are blue and the servers are red. Both the client and server are denoted by square shapes. The router has a diamond shape and has been assigned yellow color. When there are packet drops these router turn into a dark pink color. The DDoS router and the client are assigned the color black. Figure 22 represents the snapshot of these graphical animations.

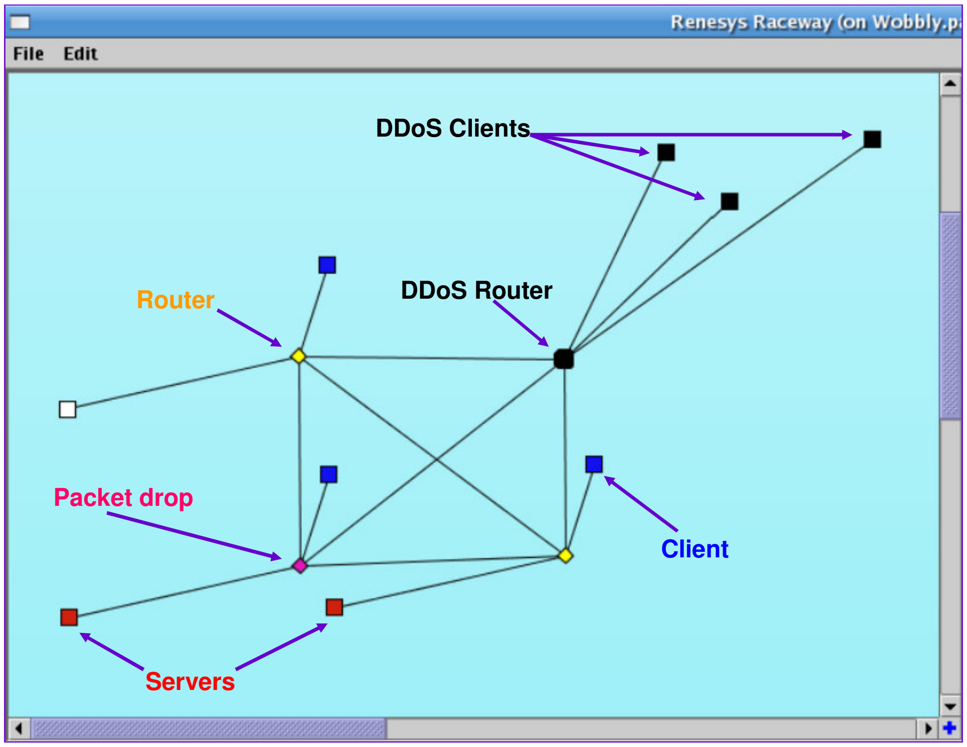


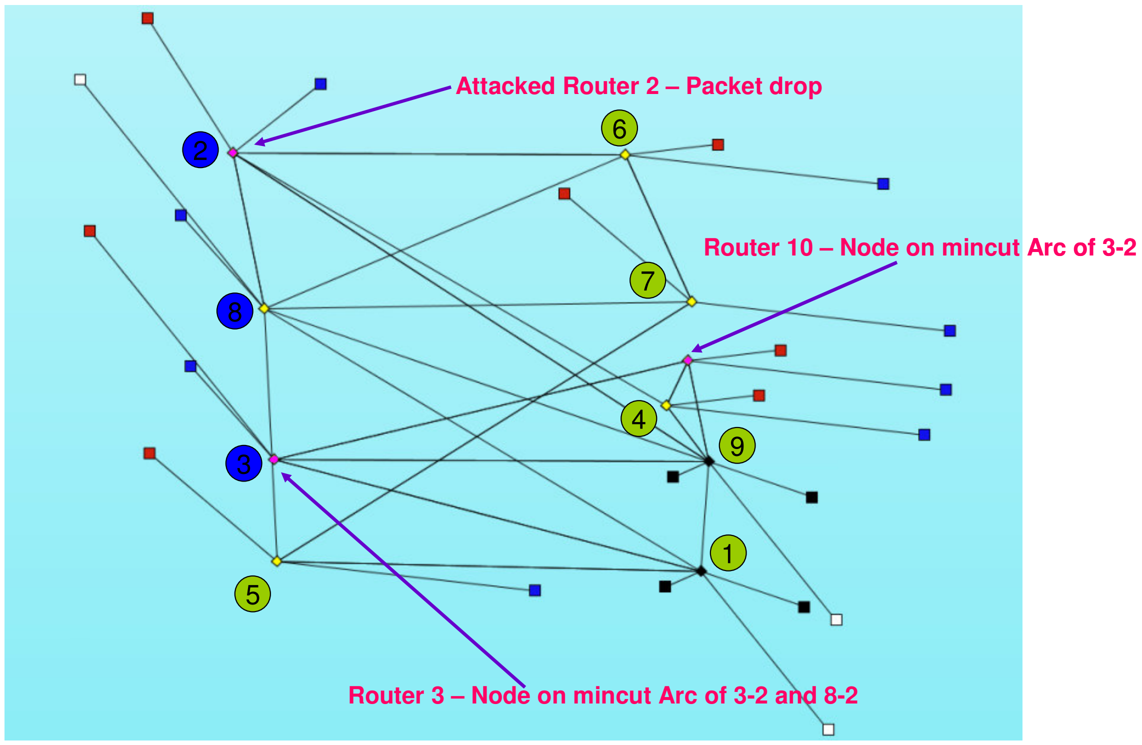
Figure 22. SSFNet animation snapshot showing the colors assigned to different hosts.

7.3 Example and Discussion

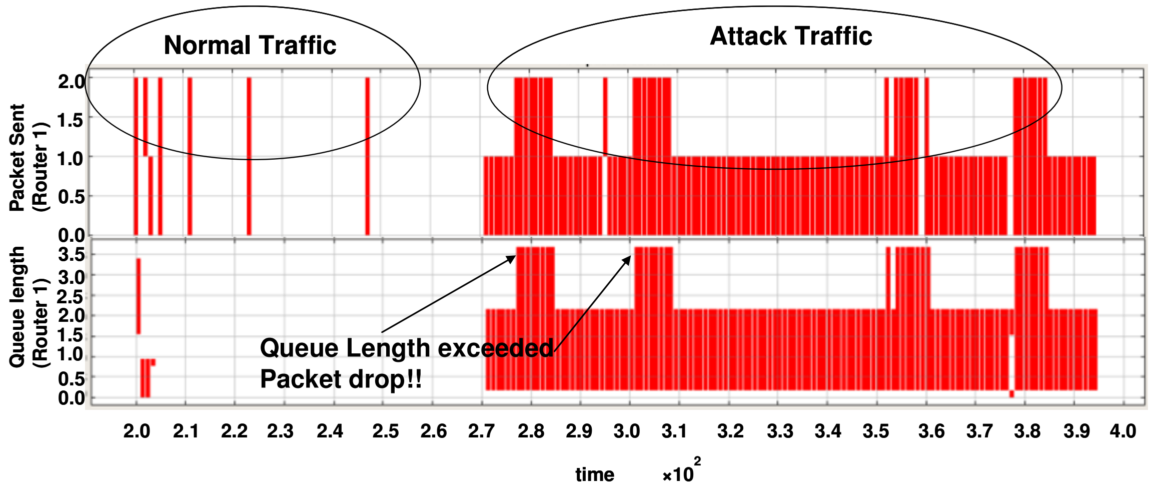
Now we perform SSFNet simulations for the example in section 7.1. The simulation time was set to 1000 seconds. The Blue configuration that was chosen randomly for the simulation was [2, 3, 8]. The minimum number of zombies needed to attack this Blue configuration was placed on physical nodes [1, 9]. The normal traffic starts at 200 seconds. The attack command is given at 250 seconds and the attack traffic starts at 270 seconds and ends at 390 seconds. Both the queue monitor and IP flow monitor were used to collect data statistics. Since we have not incorporated the use of queues for our simulations, the queue length was kept a little low and equal to 50 packets. Figure 23 is the animation snapshot of the example and the corresponding graphs of statistics collected using the queue monitor. The graphs statistics show the packets sent and the queue length at the attacked router. The requests (attack packets) are sent to the server connected to the router. But the filtering and the attack are evident at the router to which the server is attached. Figure 23.a represents the snapshot when the attack has just started on node 2. One can clearly see that three nodes → 2, 3 and 10 are pink in color i.e. there is continuous packet drop on these three nodes. On examining the min-cut sets of arcs, node 3 to node 2 have four arcs in the min-cut set – [3-1], [10-3], [9-4] and [10-4] and then if we look at the packet drops, router 3 and router 10 have packet drops along with router 2 clearly indicating the bottleneck at the arcs connecting these nodes has been exhausted. Figure 23.b shows the exceeded queue length of router 2. Similar kind of behavior is seen at router 10 and router 3, though the packet drops at router 3 are much higher. Node 8 to node 2 has one arc in the min-cut set → [3-1] signifying that packet drops should occur at node 3 as node 3-1 will be the bottleneck link. Note that this snapshot is taken at the time where only Node 2 is attacked. So the packet drop on node 3 at this time of the simulation has occurred due to it being on the min-cut arc connecting node 8 to node 2 and not due it being a Blue host.

For verification purposes zombie nodes placed were placed on physical nodes – 5, 6 and 7 that were strongly connected to the Blue sources. These nodes being very close to source do cause flooding and packet drops at the Blue sources they are close to. For example a zombie placed at node 6 causes packet drops at node 8 but fails to affect node 2 and 3. The same scenario is seen when we place a zombie on 7, it only affects node 8.

One of the reasons why 1 and 9 are good zombie places is that they are strongly connected to the nodes belonging to the min-cut and the Blue sources. Also they have much larger capacities than node 5,6 and 7 thereby causing more DDoS flow to be directed to the victims.


Attacked Router 2 – Packet drop
Router 10 – Node on mincut Arc of 3-2
Router 3 – Node on mincut Arc of 3-2 and 8-2
2
8
3
6
7
4
9
1
5

(a)



(b)

Figure 23. SSFNet simulation for Example 1 (a) Animation (b) Graphical statistics

In SSFNet nodes are represented starting with 0, so the 10 nodes are represented from 0-9 and the node so in the graphs the attacked router 2 is represented by 1.

Next we simulated the example of a large network that has 500 nodes and about 1500 arcs. In this example we required 5 zombies to disrupt Blue. SSFNet simulator is scalable and easy to use even for large networks. To prevent repetition we have just included the graphs (Figure 24) that are obtained at the attacked nodes. This is a live plot of the simulation and the Figure 24 represents the part of the live plot when the attack occurs (250-400 seconds).

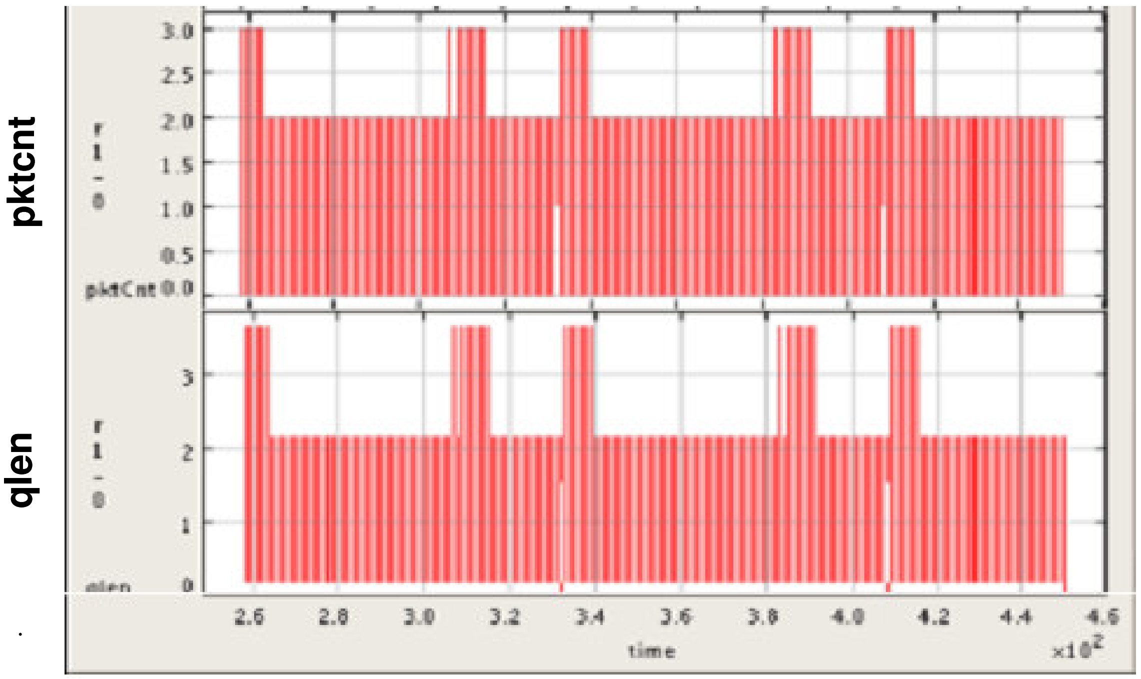


Figure 24. Graphical statistics of attack traffic of a large network.

One more observation during simulations was that the queue length had to be adjusted to get accurate results. This value varied according to the number of nodes and the arc capacities. Several rounds of simulations had to be performed to zero down on an approximate buffer length. Calculation of approximate buffer length could be added to the model.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

A generic framework for modeling network attacks is presented. With sufficient resources, it is shown that an intelligent attacker can successfully and permanently disable a network. A detailed methodology is presented where attackers with perfect knowledge of the networks they are attacking can use a modification of the well known max-flow min-cut algorithms to disable networks through packet flooding. The results show that we have been successful in quantifying the resources an attacker needs to disable a distributed application. This information can be used to build robust DDoS resistant networks.

We also set up a game between the attacker and multiple distributed applications of any enterprise. In practice, the attacker might not have sufficient resources to disrupt all the processes of an enterprise so he will try to maximize the number of processes it can disable. In reaction to this an enterprise can shift to another configuration that has not been attacked. Both the players have to determine the best process to make a move in. This problem is P-Space complete, so we have tried to solve problem using concepts of Combinatorial Game Theory and Thermographs. We have been able to provide reconfiguration strategies for the distributed application using thermostrat.

In this research it was assumed that both players have a perfect knowledge of the network. Games can be devised and studied in which knowledge of the graph is obtained over the course of time. Also we have assumed that the blue program does not have any redundancy i.e. multiple copies. Adding redundancy to the blue program will have advantages as well as disadvantages. The advantage obviously being that it will improve blue connectivity. The disadvantage being that we will require additional resources.

Further research will be focused on:

1. Introducing background traffic (with Pareto distribution) which is neither attack traffic nor Blue traffic.
2. Players not having perfect knowledge
3. Introducing redundancy to Blue
4. Cross-checking the reconfiguration strategy results obtained from the thermographs using simulations
5. Comparing the three strategies, thermostrat, hotstrat and Sentestrat
6. Calculation of approximate queue length.

The following application domains could benefit from this approach:

1. *Local Area Networks* (LANs)*:* We assume there are no zombies on local machines, but zombies exist in the larger Internet that may target processes on the LAN. This approach identifies system bottlenecks and tells the administrator if the volume of the external traffic is enough to compromise distributed processes on the LAN.
2. *Corporate Networks:* When geographically separate offices (remote locations) are connected over the Internet using a Virtual Private Network (VPN), zombies can attack the VPN traffic that travels through the global Internet. By considering the graph structure of the VPN connections between corporate controlled autonomous

systems, it is possible to create an adaptive VPN infrastructure that can tolerate DDoS attacks.

3. *Global routing problems:* Routing between autonomous systems (AS's) uses the Border Gateway Protocol, which is subject to instability in the presence of flooding DDoS attacks. Since some domains (*.edu, *.net, *.ru, …) are more likely to host zombies than others (*.mil, *.gov, …), we can analyze the AS graph structure to determine if the volume of traffic reaching sensitive BGP nodes is enough to disrupt the routing between critical agencies.

APPENDICES

MATLAB Tool Results

The results obtained on 10 physical nodes (30 arcs) and 3 Blue nodes are given below. These results include the possible Blue configurations (Blue_Placement), the number of zombies and the corresponding zombie places (Final_Zombie_List) for the following number of physical nodes (N), *EE* (Red_Con_Mat), *EV* (Red_CPU_load), number of blue nodes (Z), (BE) Blue_Con_Mat and (BV) Blue_CPU_load.

Results –Example 1:

```
N =

    10


Red_Con_Mat =

   (3,1)        1000000
   (5,1)        1000000
   (4,2)        1000000
   (6,2)        1000000
   (8,2)        1000000
   (9,2)        2000000
   (1,3)        3000000
   (9,3)        2000000
  (10,3)        2000000
   (9,4)        1000000
  (10,4)        1000000
   (1,5)        1000000
   (7,5)        1000000
   (2,6)        1000000
   (7,6)        1000000
   (8,6)        1000000
   (5,7)        2000000
   (6,7)        2000000
   (8,7)        1000000
   (1,8)        5000000
   (2,8)        1000000
   (5,8)        1000000
   (1,9)        1000000
   (2,9)        1000000
   (3,9)        1000000
   (4,9)        1000000
   (8,9)        1000000
  (10,9)        1000000
   (3,10)       2000000
   (4,10)       2000000
   (9,10)       1000000


Red_CPU_load =

     2     2     2     2     2     2     2     2     2     2


Z =

     3


Blue_Con_Mat =

   (2,1)        2000000
   (3,1)        2000000
   (1,2)        1000000
   (3,2)        1000000
   (1,3)        2000000
   (2,3)        1000000


Blue_CPU_load =

     2     2     2



Blue_Placement =

     2     1     3
     2     1     9
     2     1    10
     2     3     8
     2     3     9
     2     3    10
     2     4     3
     2     4     8
     2     4     9
     2     4    10
     2     5     3
     2     5     9
     2     5    10
     2     8     3
```

```
    2     8     9
    2     8    10
    2     9     3
    2     9     8
    2     9    10
    2    10     3
    2    10     8
    2    10     9
    3     1     2
    3     1     9
    3     1    10
    3     2     9
    3     2    10
    3     4     2
    3     4     9
    3     4    10
    3     5     9
    3     5    10
    3     8     9
    3     8    10
    3     9     2
    3     9    10
    3    10     2
    3    10     9
    9     1     2
    9     1     3
    9     1    10
    9     2     3
    9     2    10
    9     3     2
    9     3    10
    9     4     2
    9     4     3
    9     4    10
    9     5     3
    9     5    10
    9     8     3
    9     8    10
    9    10     2
    9    10     3
   10     1     2
   10     1     3
   10     1     9
   10     2     3
   10     2     9
   10     3     2
   10     3     9
   10     4     2
   10     4     3
   10     4     9
   10     5     3
   10     5     9
   10     8     3
```

```
    10     8     9
    10     9     2
    10     9     3

Final_Zombie_List = 

    [         9]
    []
    [         9]
    [1        9]
    [         1]
    [1        9]
    [1        9]
    [1        9]
    [         1]
    [1        9]
    [1        9]
    [         1]
    [1        9]
    [1        9]
    [         1]
    [1        9]
    [         1]
    [         1]
    [         1]
    [1        9]
    [1        9]
    [         1]
    [         9]
    []
    []
    [         1]
    [1        9]
    [1        9]
    []
    []
    [         1]
    [1        9]
    [         1]
    [1        9]
    [         1]
    []
    [1        9]
    []
    []
    []
    []
    [         1]
    [         1]
    [         1]
    []
    [         1]
    []
```

```
     []
     [         1]
     [         1]
     [         1]
     [         1]
     [         1]
     []
     [         9]
     []
     []
     [1        9]
     [         1]
     [1        9]
     []
     [1        9]
     []
     []
     [1        9]
     [         1]
     [1        9]
     [         1]
     [         1]
     []
```

Appendix B

SSFNet Simulations

SSFNet [66] is a network simulator developed by the SSFNet project. The simulator consists of three major parts, SSF, DML, and SSFNet. It is a collection of Java SSF-based components for modeling and simulation of Internet protocols and networks at and above the IP packet level of detail. Domain Modeling Language (DML) is a public-domain standard for attribute databases for model configuration and verification. It supports extensibility, inheritance and substitution of attributes. One of the main reasons for using SSFNet in our simulations is that it is scalable. Due to this one can simulate large scale networks with ease. The approach had to be tested on a large scale network as an attacker can easily deploy thousands or millions of zombies.

B.1 Using SSFNet

In order to use SSFNet (latest version 2.0) one needs to get install SSFNet and JDK1.3. To run SSFNet, we first need to set the classpaths to the jar files in the directory ssfnet/lib and the parent directory of any of the classpaths required. To configure a network one needs to write a DML file. It has a very simple syntax. Informally speaking, it's a list of attributes. Each attribute is a key-value pair. The attribute key is basically an identifier. The attribute value can be a number or a string. To learn DML for SSFNet, the best way would be to see the examples in the ssfnet directory or in [66]. Specially [66] goes through the meaning of each line in the DML file. This includes configuring a Net, a

Host or a Router, the links connecting the hosts and NIC's. The host can be either a server or a client. In our directed graph, each node should have the capability to both send and receive. So to incorporate this in SSFNet we have made each node to have three nodes in it i.e. a server host, a client host and a router. The server and the client are connected to the router with bandwidth that is 100 times or much more than the bandwidth connecting the two routers. For e.g. if we have two physical nodes 1 and 2 and the connectivity matrix *EE* is given by;

$$EE = \begin{bmatrix} 0 & 3 \\ 1 & 0 \end{bmatrix}$$

Then in SSFNet we will configure it as shown in Figure 25. Note that the bidirectional link between the router and the each of the host has much more capacity than it requires. This is done in order to simulate a single node that can act both as a server and a client.

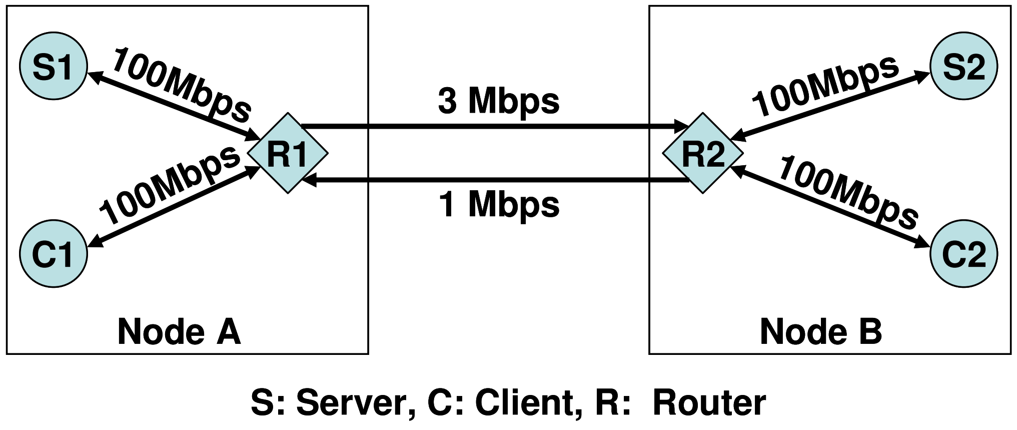


Figure 25. SSFNet representation of two Nodes A and B.

To simulate the distributed application Blue, TCP connections were set up between the physical nodes of Blue configuration (Blue_Placement). Any one Blue configuration was picked up at random. Zombies were placed on the physical nodes in the SSFNet network simulator according to the Zombie Placement (Final_Zombie_List) and the rate was kept equal to RT.

DDoS Clients: SSFNet has a DDoS package that provides facilities to set up a DDoS attack. DDoSSession and DDoSSessionRand are two protocol session implementations of DDoS SYN packets. DDoSSessionRand has more advanced features than DDoSSession and also configures the attributes for the master and agents automatically. For one of the clients (nodes) of a DDoS client we have to have 3 clients incorporated in a single node so that we can have an attacker, master and the zombie. For all the other DDoS clients the same attacker and master can issue the attack command.

Background Traffic: Some background traffic – UDP was introduced in the simulation between nodes that are neither zombies nor host any Blue nodes to give the simulation a real approach.

Unidirectional Links: Another problem we had to deal with during simulations was that unidirectional links had not yet been implemented in SSFNet. For our analysis we could not have bidirectional links as we use a directed graph structure. So we had to specify the bandwidth for each link separately. The bidirectional links would mean redundant bandwidth available which could lead to incorrect results. In order to incorporate simplex links in SSFNet, we set different bit rates at both ends of the link. So say if we have a bidirectional link between node *A* and node *B*, we can turn it in a unidirectional link from A to *B* by setting the bit rate at interface of *B* to a very low value like 1. In this way the packets will never be sent on the bidirectional link from *B* to *A* but packets from *A* to be *B* will be readily transferred.

In order to run the DML file, one has to give the following command;

```
java SSF.Net.Net <requested simulation end time in seconds> <Filename>.dml
```

Monitoring: SSFNet supports efficient multi-point network monitoring infrastructure for collection of streaming and sampled data from many Monitors. The package SSF.Util.Streams together with SSFNet class SSF.OS.ProbeSession provides monitoring facilities. Monitoring in SSFNet is of two types, *(i)* Queue Monitoring *(ii)* IP Netflow Monitoring. Queue Monitoring is done at the session level, where packets can be captured at the router queues and analyzed for queue lengths, packet drops etc. IP Netflow Monitoring is used to monitor flows at routers, IP packet dumps on interfaces and links, and analyze end to end flow. Flows can be further refined by additional attributes like protocol number, and can be aggregated in a variety of ways like by source and/or destination network prefix, for analysis purposes. For monitoring purposes the user has to specify the name of the stream and data file that collects the data for the particular stream. This data file can then be used to plot graphs for analysis or for animation of the network. For more information readers are referred to [66] or the ssfnet/examples/queueMonitorDemo directory. [67] gives additional information on the IP Netflow monitoring along with data format for the data file generated.

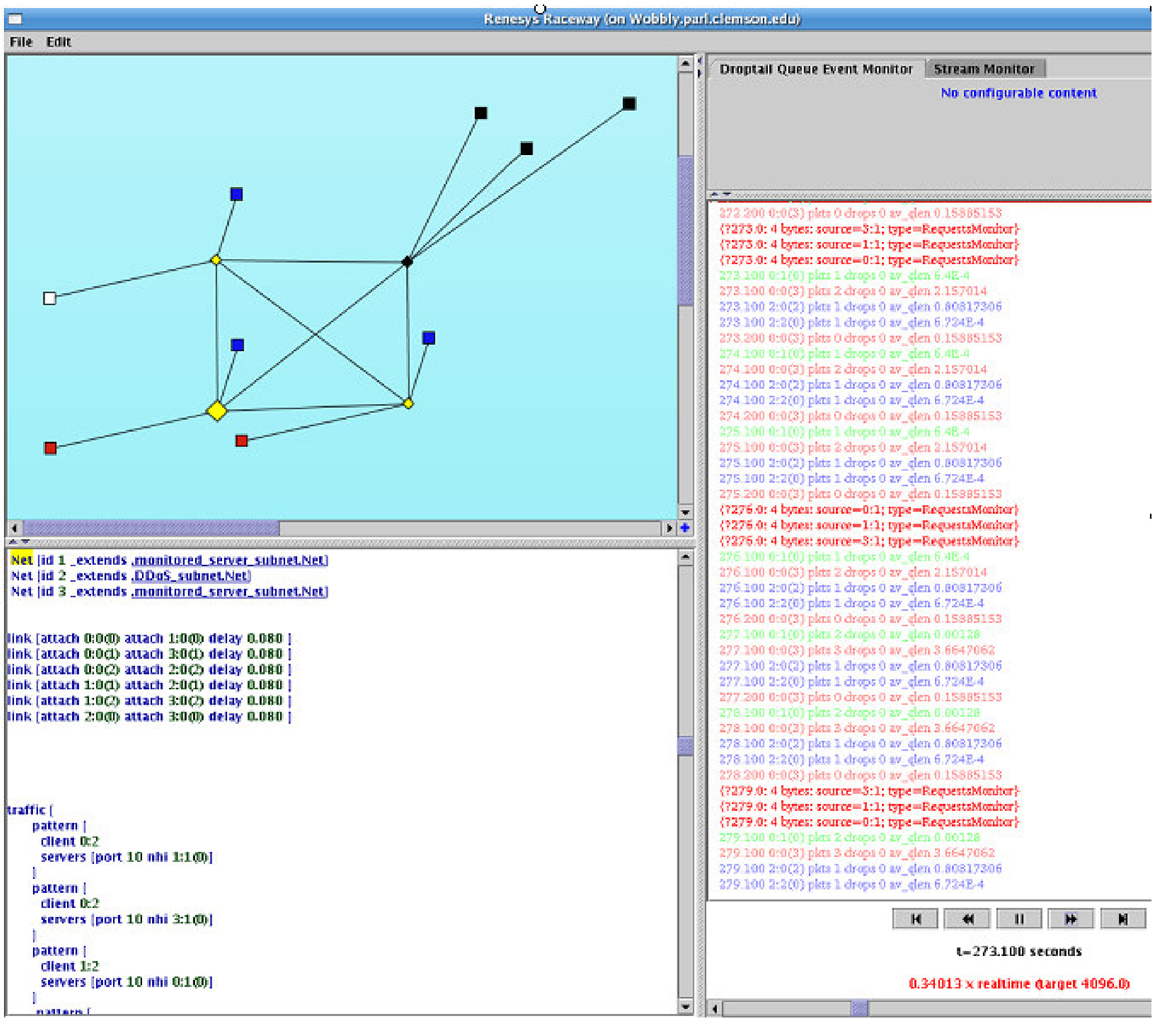
Graphical representation: In SSFNet, there is provision for graphical representation and animation in the form of Raceway Viewer [66]. Raceway Views provides the framework for displaying network topology. The DML network configurations used by Raceway Views are the same as used in SSFNet, but in addition to logical configuration attributes for Nets, hosts, routers and links they include new graphics attributes specifying the locations, transformations, and rendering styles for Nets and for individual network elements. If no graphical attributes are supplied, SSFNet will assign the attributes to the nodes and the links randomly. One can fully customize in DML the rendering style, including link colors and stroke widths, icon selection and fill color, size and orientation for hosts, routers etc.

Network Animation: The stream generated from the simulation can also be played using a player written in java. Though there are some examples of the different players in the animation/players directory, these are written with specific DML files in mind. For the simulations presented here, a player was written that was built on the droptail player example in the animations directory.

Figure 26 represents a snapshot of the Raceway viewer for three blue nodes and four physical nodes. The Raceway Viewer has three windows/components. The first is the network renderer. It shows the network topology using the network configuration DML files. One can change the background color, and the position of the network elements. The second record animation component is the running display of a textual representation of network records in the format - timestamp, originating network element address, record type, total record length in bytes, followed by record-type-specific measurement data bytes. These records can be examined for drill-down analysis. The third component is a set of VCR-like control buttons (Play, Stop, Rewind, Forward, etc) that are present on the sidebar. The user can add or remove buttons according to his convenience.

To run the animations and start Raceway Viewer the following command should be given;

java RacewayViewer <Filename>.dml -s <DataFile>.dat –p <Player>.java -start 30 -stop 10000 (optional - if the start/stop options are provided, they define the timestamp range for records to be played; otherwise, the entire stream is played).


Renesys Raceway (on Wobbly.pari.clemson.edu)
File
Edit
Droptail Queue Event Monitor
Stream Monitor
No configurable content

```
Net [id 1 _extends .monitored_server_subnet.Net]
Net [id 2 _extends .DDoS_subnet.Net]
Net [id 3 _extends .monitored_server_subnet.Net]


link [attach 0:0(0) attach 1:0(0) delay 0.080 ]
link [attach 0:0(1) attach 3:0(1) delay 0.080 ]
link [attach 0:0(2) attach 2:0(2) delay 0.080 ]
link [attach 1:0(1) attach 2:0(1) delay 0.080 ]
link [attach 1:0(2) attach 3:0(2) delay 0.080 ]
link [attach 2:0(0) attach 3:0(0) delay 0.080 ]




traffic [
    pattern [
      client 0:2
      servers [port 10 nhi 1:1(0)]
    ]
    pattern [
      client 0:2
      servers [port 10 nhi 3:1(0)]
    ]
    pattern [
      client 1:2
      servers [port 10 nhi 0:1(0)]
    ]
    pattern [
```

```
272.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
{?273.0: 4 bytes: source=3:1; type=RequestsMonitor}
{?273.0: 4 bytes: source=1:1; type=RequestsMonitor}
{?273.0: 4 bytes: source=0:1; type=RequestsMonitor}
273.100 0:1(0) pkts 1 drops 0 av_qlen 6.4E-4
273.100 0:0(3) pkts 2 drops 0 av_qlen 2.157014
273.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
273.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
273.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
274.100 0:1(0) pkts 1 drops 0 av_qlen 6.4E-4
274.100 0:0(3) pkts 2 drops 0 av_qlen 2.157014
274.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
274.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
274.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
275.100 0:1(0) pkts 1 drops 0 av_qlen 6.4E-4
275.100 0:0(3) pkts 2 drops 0 av_qlen 2.157014
275.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
275.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
275.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
{?276.0: 4 bytes: source=0:1; type=RequestsMonitor}
{?276.0: 4 bytes: source=1:1; type=RequestsMonitor}
{?276.0: 4 bytes: source=3:1; type=RequestsMonitor}
276.100 0:1(0) pkts 1 drops 0 av_qlen 6.4E-4
276.100 0:0(3) pkts 2 drops 0 av_qlen 2.157014
276.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
276.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
276.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
277.100 0:1(0) pkts 2 drops 0 av_qlen 0.00128
277.100 0:0(3) pkts 3 drops 0 av_qlen 3.6647062
277.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
277.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
277.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
278.100 0:1(0) pkts 2 drops 0 av_qlen 0.00128
278.100 0:0(3) pkts 3 drops 0 av_qlen 3.6647062
278.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
278.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
278.200 0:0(3) pkts 0 drops 0 av_qlen 0.15885153
{?279.0: 4 bytes: source=3:1; type=RequestsMonitor}
{?279.0: 4 bytes: source=1:1; type=RequestsMonitor}
{?279.0: 4 bytes: source=0:1; type=RequestsMonitor}
279.100 0:1(0) pkts 2 drops 0 av_qlen 0.00128
279.100 0:0(3) pkts 3 drops 0 av_qlen 3.6647062
279.100 2:0(2) pkts 1 drops 0 av_qlen 0.80817306
279.100 2:2(0) pkts 1 drops 0 av_qlen 6.724E-4
```

```
t=273.100 seconds

0.34013 x realtime (target 4096.0)
```

Figure 26. SSFNet snapshot showing the three components of graphical representation.

REFERENCES

[1] In Estonia, what may be the first war in cyberspace, May 28, 2007, International Herald Tribune [Online].
Available: http://www.iht.com/articles/2007/05/28/business/cyberwar.php

[2] G. Carl, G. Kesidis, R. Brooks and S. Rai, "Denial-of-Service Attack-Detection Techniques," *IEEE Internet Computing*, vol. 10, nos. 1, pp. 82-89, Jan. 2006.

[3] J. Mirkovic, J. Martin, and P. Reiher, "A taxonomy of DDoS attacks and DDoS defense mechanisms," in *ACM CCR*, April 2004.

[4] R. R. Brooks, *Disruptive Security Technologies with Mobile Code and Peer-to Peer Networks*, Boca Raton, FL: CRC Press, 2005.

[5] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows*, Prentice Hall, Upper Saddle River, NJ, 1993.

[6] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your mathematical plays Volume 1: Games in General,* Academic Press, New York, 1982.

[7] E. R. Berlekamp, "The Economist's View of Combinatorial Games," in It Nowakowski (Ed.), *Games of No Chance,* MSRI Publications, Vol. 29, Cambridge University Press, Cambridge, 1996, pp. 365-405.

[8] A. Garg and A. L. N. Reddy. "Mitigation of DoS attacks through QoS Regulation" in *Proceedings of IWQOS workshop*, May 2002.

[9] W. Stallings, *Network and Internetwork Security,* Prentice Hall, Upper Saddle River, NJ, 1995.

[10] CERT Coordination Center, Denial of Service Attacks [Online]. Available: http://www.cert.org/tech_tips/denial_of_service.html.

[11] R. H. Anderson, P. M. Feldman, S. Gerwehr, B. K. Houghton, R. Mesic, J. Pinder, J. Rothenberg, J.R. Chiesa, "Securing the U.S. Defense Information Infrastructure: A Proposed Approach," *RAND Corporation*, Santa Monica, 1999 [Online]. Available: http://www.rand.org/publications/MR/MR993

[12] Frontline, "Cyber War!" [Online].
Available: http://www.pbs.org/wgbh/pages/frontline/shows/cyberwar/warnings/

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

[14] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, “Distributed Denial of Service Attacks,” in *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2275-2280, Nashville, TN, USA, October 2000.

[15] Kevin, J., Weaver, G. M., Long, N., and Thomas, R, “Trends in denial of service attack technology,” Technical report, CERT Coordination Center, Carnegie Mellon University, October, 2001.

[16] J. Allen, “Governing for Enterprise Security,” *CERT*, 2005.

[17] ProCurve Networking by HP, “Protecting the Extended Enterprise Network: Security Strategies and Solutions from ProCurve Networking,” in ProCurve Networking, HP Innovation [Online].
Available: http://www.hp.com/rnd/pdfs/securitypaperfinal.pdf

[18] D Piscitello, and L Phifer, “Best Practices for Securing Enterprise Networks,” in *Business Communications Review*, 2002 [Online].
Available: http://findarticles.com/p/articles/mi_hb5084/is_200212/ai_n18468855

[19] Control Scan, “What is Network Security: A Guide to the Importance of Keeping your Company’s Network Secure,” in *Network Security, Control Scan* [Online].
Available:http://www.controlscan.com/corp/AGuidetotheImportanceofKeepingyourCom panysNetworkSecure.pdf

[20] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*, Wiley, NY, 2004.

[21] K. D. Mintick and W. L. Simon, *The Art of Deception: Controlling the Human Element of Security*, Wiley Publishing, Indianapolis, IN, 2002.

[22] K. Wong, “The Hackers and Computer Crime,” in *Proceedings Securicom 1986 4th World Congress on Computer and Communications Security and Protection*, pp. 11-26, Paris, March 1986.

[23] Better Business Bureau, “New Research Shows Identity Fraud Growth Is Contained and Consumers Have More Control Than They Think,”

[24] J. D. Howard, T. A. Longstaff, *A Common Language for Computer Security Incidents*, Sandia Report, SAND98-8867.

[25] R. R. Brooks, “Mobile code paradigms and security issues,” *IEEE Internet Computing*, vol. 8, no. 3, pp. 54-59, May/June 2004.

[26] Federation of American Scientists, “Countering Industrial Espionage in the Post-cold-war Era” *Congressional Record* excerpt, 6/24/92

[27] BBC, “China's spies come out from the cold,” 07/22/05 [Online]. Available: (http://news.bbc.co.uk/2/hi/asia-pacific/4704691.stm) Last visited 12/11/2006.

[28] R. C. Brackney, and R. H. Anderson, *Understanding the Insider Threat Proceedings of a Workshop*, Rand Corporation Conference Proceedings, March 2004.

[29] M. Keeney, E. Kowalski, D. Cappelli, A. Moore, T. Shimeall, and S. Rogers, Insider Threat Study: *Computer System Sabotage in Critical Infrastructure Sectors,* United States Secret Service and Carnegie-Mellon CERT, May 2005 [Online]. Available: (http://www.cert.org/archive/pdf/insidercross051105.pdf) Last visited 12/12/2006.

[30] K. Lam, D. LeBlanc, and B. Smith, *Assessing Network Security,* Microsoft Press, Redmond, WA 2004.

[31] B. Schneier, “Attack Trees Modeling Security Threats,” *Dr. Dobb’s Journal*, December 1999. Available: http://www.counterpane.com/attacktrees-ddj-ft.html.

[32] B. Harris, and R. Hunt, “TCP/IP security threats and attack methods”, *Computer Communications*, vol. 22, no. 10, 25 June 1999, pp. 885-897.

[33] R. W. Stevens, *TCP/IP Illustrated*, vol’s 1, 2, and 3, Addison-Wesley, Reading, MA, 1994.

[34] S. Dietrich, N. Long, and D. Dittrich, “Analyzing Distributed Denial of Service Attack Tools: The Shaft Case”, in *Proceedings of the LISA 2000 System Administration Conference*, December 2000, New Orleans, LA.

[35] T. Peng, C. Leckie, and K. Ramamohanarao, “Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems,” *ACM Computing Surveys,* vol. 39, no. 1, article 3, April 2007.

[36] L. Garber, "Denial of Service Attacks Rip the Internet," *Computer*, vol. 33, no.4, pp. 12-17, April 2000 [Online].
Available: (http://www.fas.org/irp/congress/1992_cr/s920624-spy.htm) Last visited 12/12/2006.

[37] R. Chen, J. Park, and R. Marchany, "A Divide-and-Conquer Strategy for Thwarting Distributed Denial-of-Service Attacks," *IEEE Transactions on Parallel and Distributed Systems,* vol 18, issue 5, May 2007, pp 577-588D.

[38] P. Owezarski, "On the impact of DoS attacks on internet traffic characteristics and QoS," in *ICCCN '05 Proceedings of the 14th International Conference on Computer Communications and Networks*, pp. 269–274, LAAS-CNRS, Toulouse, France, IEEE, October 2005.

[39] D. K.Y. Yau, J.C.S. Lui, and F. Liang, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," in *Proceedings of IEEE IWQoS*, May 2002.

[40] J.Jung, B.Krishnamurthy, and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and WebSites," in *Proceedings of International World Wide Web Conference*, 2002.

[41] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the Source," in *Proceedings of the ICNP 2002*, November 2002.

[42] N. Aaaraj, S. Itani, and D. Abdelahad, "Neighbor stranger discrimination: a new defense mechanism against Internet DDOS attacks," The 3rd ACS/IEEE International Conference on Computer Systems and Application, pp. 95-102, 2005.

[43] K. Park and H. Lee, "On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack", in *Proceedings of IEEE INFOCOM 2001*, March 2001.

[44] S. Chen, and Q. Song, "Perimeter-based Defense against High Bandwidth DDoS Attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, No. 6, June 2005.

[45] C. Jin, H. Wang, and K. G. Shin, "Hop-count Filtering: An Effective Defense against Spoofed Traffic," in *ACM Conference on Computer and Communications Security (CCS)*, Oct. 2003.

[46] S. Zhang and P. Dasgupta, "Denying Denial of Service Attacks: A Router Based Solution," *The 2003 International Conference on Internet* Computing, June 2003.

[47] H. L. Flanagan, "Egress filtering – keeping the Internet safe from your systems," *GSEC Practical Assignment Version 1.2c*, April 2001.

[48] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet denial-of-service with capabilities," in *2nd ACM Hotnets Workshop*, Cambridge, MA, Nov. 2003.

[49] K. Park, and H. Lee, “On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack,” *IEEE INFOCOM*, vol. 1, pp. 338-347, 2001.

[50] R. Chang, “Defending against flooding-based, distributed denial-of-service attacks: a tutorial,” *IEEE Communications Magazine*, 40(10), 2002.

[51] A. Habib, M. Hefeeda, and B. Bhargava, “Detecting Service Violations and DoS Attacks,” in *NDSS Conference Proceedings*, Internet Society, 2003.

[52] J. O. Royset, and R. K. Wood, “Solving the Bi-Objective Maximum-Flow Network Interdiction Problem,” in *INFORMS Journal of Computing*, vol. 19, No. 2, pp. 175-184, Spring 2007.

[53] C. Thompson, “Scaling Object Services Architectures to the Internet: Internet Tools Survey – Quality of Service,” DARPA Project Description, 1998 [Online].
Available: http://www.objs.com/OSA/Final-Report.html

[54] J. Clausen, “Branch and Bound Algorithms - Principles and Examples,” Department of Computer Science, University Copenhagen, [Online].
Available: http://www.imm.dtu.dk/~jha/

[55] E. L. Lawler and D. E. Wood, “Branch and bound methods: A survey,” *Operations Research*, vol. 14, pp. 699–719, 1966.

[56] L. J. Yedwab, “On playing well in a sum of games,” M.S. Thesis, MIT, 1985, MIT/LCS/TR-348.

[57] D. E. Knuth, *Surreal Numbers*, Addison-Wesley, Reading, MA, January, 1974.

[58] C. Tondering, “Surreal Numbers - An Introduction,” Version 1.5, January 2005 [Online].
Available: http://www.tondering.dk/claus/surreal.htmlv (last visited 08/10/2006).

[59] J. H. Conway, *On Numbers and Games*, AK Peters, LTD, 2000.

[60] E. D. Demaine, “Playing games with algorithms: Algorithmic combinatorial game theory,” in *Proceedings 26th Symposium on Mathematical Foundations in Computer Science*, vol. 2136 of Lecture Notes in Computer Science, pp. 18-32, August 2001.

[61] A. N. Siegel, “Loopy Games and Computation,” PhD thesis, University of California at Berkeley, 2005.

[62] M. Muller, E. Berlekamp, and B. Spight, “Generalized Thermography: Algorithms, Implementation, and Applications to Go endgames,” Technical Report 96-030, ICSI Berkeley, 1996.

[63] E. Mendelson, *Introducing Game Theory and Its Applications*, Chapman & Hall / CRC, Boca Raton, FLA 2004.

[64] B. C. A. Milvang-Jensen, “Combinatorial Games, Theory and Applications,” Thesis, IT University of Copenhagen, 2000.

[65] E. Berlekamp and D. Wolfe, *Mathematical Go or Chilling Gets the Last Point*, A K Peters, Ltd, Wellesley, MA, 1994.

[66] Open source SSFNet simulator [Online].
Available: http://www.ssfnet.org/homePage.html

[67] SSF.OS.Netflow examples for Netflow monitoring [Online].
Available:http://www.ssfnet.org/javadoc/SSF/OS/NetFlow/package-summary.html